Explorative Graph Visualization

Dissertation

zur

Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

vorgelegt von

Dipl. Inf. Hans-Jörg Schulz, geb. am 26. Februar 1979 in Rostock

aus Bad Doberan

Rostock, 1. April 2010

urn:nbn:de:gbv:28-diss2010-0090-9

Principal Advisor: Prof. Dr.-Ing. habil. Heidrun Schumann
University of Rostock, Germany

External Reviewers: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Keith Andrews
Technical University Graz, Austria

Dr. James Abello, Research Professor
DIMACS Institute, Rutgers University, USA

Date of Defense: 09-JUN-2010

Abstract

Network structures (graphs) have become a natural part of everyday life and their analysis helps to gain an understanding of their inherent structure and the real-world aspects thereby expressed. The exploration of graphs is largely supported and driven by visual means. The aim of this thesis is to give a comprehensive view on the problems associated with these visual means and to detail concrete solution approaches for them. This is done on all three levels involved: the data level, the representation level, and the task level.

In a first step, the intricate dependencies on the representation level alone are discussed for the case of implicit tree visualizations. With the awareness of the representation issues involved, a second step takes the characteristics on data level into account, which are most importantly graph size and graph type. Their influence on the visualization design are discussed specifically for large trees and bipartite graphs. Finally, with bringing in the task level, the entire exploration workflow from computational preprocessing to the interaction with the visualization can be captured and thus discussed in terms of its consequences for the visualization design. This includes issues of necessary confirmative elements in the exploratory analysis, as well as selecting appropriate data sets from a heterogeneous data pool for an analysis.

Concrete visualization techniques are introduced to underline the value of this comprehensive discussion for supporting explorative graph visualization.

Keywords: graph visualization, explorative visualization, graph exploration

CR Classification: H.5.0, I.3.8, J.3

Zusammenfassung

Netzwerkstrukturen (Graphen) sind heutzutage weitverbreitet. Ihre Untersuchung dient dazu, ein besseres Verständnis ihrer Struktur und der durch sie modellierten realen Aspekte zu gewinnen. Die Exploration solcher Netzwerke wird zumeist mit Visualisierungstechniken unterstützt. Ziel dieser Arbeit ist es, einen Überblick über die Probleme dieser Visualisierungen zu geben und konkrete Lösungsansätze aufzuzeigen. Da diese Visualisierungen im Wesentlichen durch die drei Faktoren Repräsentation, Daten und Aufgaben beeinflußt werden, wird die Diskussion in drei Schritten geführt.

In einem ersten Schritt werden die miteinander verflochtenen Probleme der Repräsentationsebene am Beispiel der impliziten Baumrepräsentationen diskutiert. Zu den dabei identifizierten Herausforderungen kommen dann in einem zweiten Schritt die Eigenschaften der Daten und deren Auswirkungen auf die Repräsentation hinzu. Dies sind hauptsächlich die Größe eines Graphen und auch dessen Typ. Beispielhaft werden diese Aspekte für die Darstellung großer Bäume und bipartiter Graphen diskutiert. Im letzten Schritt werden dann die Aufgaben betrachtet und es werden die sich ergebenden Vorteile einer umfassenden Modellierung aller drei Einflußfaktoren (Repräsentation, Daten und Aufgaben) auch über den Rahmen der Exploration hinaus am Beispiel einer automatischen Auswahl von Daten für eine gegebene Aufgabe herausgestellt.

Neue Visualisierungstechniken werden in jedem Schritt eingeführt, um den Nutzen der geführten Diskussion für die explorative Graphvisualisierung am konkreten Beispiel zu belegen.

Schlagwörter: Graph Visualisierung, Explorative Visualisierung, Graph Exploration

CR Klassifikation: H.5.0, I.3.8, J.3

Contents

List of Figures

List of Tables

Acknowledgements

My sincerest thanks go to all the people who advised and encouraged me while conducting this thesis. First and foremost, a special thank you goes to my supervisor Heidi Schumann. Her constructive criticism of my work and her confidence in my ideas guided me the long way from my first literature research to the completion of this thesis. I am also grateful to James Abello for introducing me to his very own perspective on graph visualization, as well as inviting me for a number of visits at DIMACS, Rutgers University. I also want to thank Keith Andrews from the Technical University of Graz for the uncomplicated interaction and his acceptance to review my thesis only a few month in beforehand. In addition to my advisors and reviewers, I want to express my gratitude to Christian Tominski and Frank van Ham, with whom I shared many ideas and had lively discussions about graphs, universe, and everything. Then there are my colleagues and friends from the Computer Graphics and Visual Computing Group, as well as from the GRK dIEM oSiRiS – thanks for your support! Furthermore, I thank my student Steffen Hadlak whose code wizardry enabled me to bring many of my ideas to life. Lastly, and most importantly, I want to thank my family, my girlfriend, and all friends for their support. This thesis could not have been written without you.

The work on this thesis was financially supported by the Evangelisches Studienwerk e.V. Villigst and the interdisciplinary DFG Graduiertenkolleg dIEM oSiRiS. Research visits to DIMACS, Rutgers University were kindly funded by DIMACS and DyDAn, a research visit to the Visual Communications Lab, IBM Research was funded by IBM.

Chapter 1

Introduction

Networks have become commonplace in main stream culture and are no longer understood by engineers and scientists only. For instance, social networks, 50 years ago just known to a few sociologists, are nowadays an integral part of most peoples' online lives. With the rising awareness of networks, their spread to numerous other fields, and their ever growing size, the need for computational aids in comprehending them and reasoning with them arose. And even though a vast number of such aids are available by now, it still remains challenging to overlook and understand the subtle dependencies in large networks, as the cascading power failures in the US energy network illustrate.

In general, when one wants to analyze data (in this case a network) in order to gain an understanding of its inherent structure, one passes through two stages [Beh97]:

1. **Exploration**, which is an undirected search for characteristic features of the network in order to generate hypotheses about it.

2. **Confirmation**, which is a directed, often exhaustive search for features of the network in order to verify or falsify preconceived hypotheses.

So, as the exploration phase is more about a familiarization with the network and the discovery of its features, it is the confirmation as a second step which ensures that the observed features are indeed distinctive and persistent. The methods used for both phases are usually of graphical or algorithmic nature, with an emphasis on the graphical methods for the exploration and on the algorithmic methods for the confirmation. Yet both of these methodological approaches are far from being disjoint as a graphical representation can hardly be obtained without a bit of algorithmic preprocessing. And on the other hand, the results of an algorithm are still displayed best using a chart, a diagram, or some other graphical means. But the distinction between graphics and algorithmics and their principal application to exploration and confirmation respectively still remains true and reasonable [**?**]:

- **Graphical methods** enable an analyst to quickly determine regularities and deviations from them as possible characteristic network features, as they speak first and foremost to the visual system and allow to achieve fuzzy objectives like an undirected search by pure visual inspection.

- **Algorithmic methods** are better suited for concrete objectives like a directed search in which visually observed network characteristics are systematically sought out to back up the observation.

This thesis concerns itself with the first of both stages: the exploration using graphical methods as an effective and solid foundation for a subsequent (algorithmic) confirmation as a second step of analysis. It does so not only for networks, but for the mathematical abstract concept of graphs – a generalization under which all kinds of different structures including networks can be subsumed.

1.1 Motivation and Problem Statement

Using visualization as a means for exploratory graph analysis can be as intricate as it can be beneficial. On the way from the data to its visual representation and finally from the interaction with this representation to a (better) understanding of the data, there are a number of caveats to overcome. Many of these have already been identified in general and thus made explicit research questions by various authors [Jer99, AS04, Che05]. Yet, the mutual influence of these aspects onto one another are rarely considered in the literature. Concrete examples for such specific challenges on data, representation, and task level are:

- **data level** – large graphs, scaling up to millions of nodes and edges ***(quantity)***, graphs of certain types, such as planar graphs or bipartite graphs ***(diversity)***
- **representation level** – well laid out, meaningful visualizations ***(readability)*** on limited screen space ***(space efficiency)***, adaptable visualizations that facilitate exploration with interactive frame rates ***(interactivity)***
- **task level** – initially fuzzy tasks with uncertain analysis goals, as a clear objective will not emerge until the exploration is in progress, as well as tasks changing midways if intermediate findings point at more promising or more urgent paths of analysis ***(variability)***

Most of the existing literature on exploratory graph analysis concerns itself with the challenges of the representation level, discussing layout problems and interaction methods. The works on the data level are mainly concerned with handling large graphs, e.g., through clustering algorithms. On the task level, there are currently not more than a handful of publications, giving a first overview of tasks related to exploratory graph analysis. This thesis focuses on the representation level, too. But at the same time, it also discusses the data and task level and their implications for the representation. Thinking these implications through is the fundamental challenge of exploratory graph visualization, because they may partly depend or contradict each other, amplify or cancel each other out.

Hence, the aim of this thesis is to discuss these challenges on data, representation, and task level, as well as their solution approaches with a specific emphasis on their implications for the representation level. This is done concretely for several scenarios

posing different challenges through their requirements and constraints. Emphasizing the applicability of the underlying design principles identified in this discussion, this thesis utilizes them for a number of applications, effectively generating specifically tailored, novel graph visualization techniques that facilitate exploratory visual analysis.

1.2 Fundamental Approaches and Results

As a starting point, a short compilation of the known building blocks of explorative graph visualization is given. This includes the different kinds of graphs (on data level), the different graph visualization approaches (on representation level), the different explorative tasks (on task level), as well as their combinations – which kind of visualization to use for which kind of graph, and which kind of task to perform on which kind of visualization. This short overview is partially based on the publication *"Visualizing Graphs – A Generalized View"*, which appeared in the Proceedings of the International Conference Information Visualisation 2006. With this foundation at hand, the problems and challenges of explorative graph visualization are identified and related to one another. Examples of existing solution approaches are given and the concrete problem instances dealt with in this thesis are motivated and specified. These are shortly described in the following.

The first part of the discussion concentrates solely on the *representation level* by targeting a problem that comes as part of the solution: the multitude of available visualizations and their many possible parametrizations demand decisions on a number of interlinked and hard to overlook design issues in order to actually generate tailor-made visualizations. For the confined field of implicit tree visualizations, this thesis is raising awareness for this problem and devises a comprehensive design space towards that end. It allows a visualization designer to quickly develop new and adapt existing implicit tree visualization techniques to suit changing demands. This solution approach is not new and parametrization of visualization techniques is a well known approach. Yet, the scope of this design space is beyond the parametrization commonly employed, as it incorporates dozens of visualizations which are known techniques in their own right, effectively capturing an entire class of tree visualizations including known and still unknown techniques likewise. To get a practical grip on the numerous visualization designs described by the design space, it is implemented as a software toolkit that can be used for rapid prototyping of implicit tree visualizations. Its usefulness is illustrated by deriving a number of novel visualization prototypes. The design space, as well as the novel prototypes are described in the publication *"The Design Space of Implicit Hierarchy Visualization: A Survey"*, which is accepted with minor revisions for the IEEE Transactions on Visualization and Computer Graphics.

The second part of the discussion brings in the *data level* and emphasizes the importance of the input graph's characteristics for the representation. The main aspects to be considered in this regard are graph size and type. For very large graphs, the representational property of space-efficiency stands at the basis of a scalable layout design. The bene-

fits of such a space-efficient layout as well as the concrete design considerations that go along with dense visualizations are examined for the concrete case of visualizing large trees. To that end, a newly developed solution approach featuring the point primitive as the smallest possible node representation and applying a hierarchical placement scheme is devised, evaluated and related with respect to other scalable tree visualizations, and finally extended and adapted to different tree properties and drawing areas. This point-based tree visualization has been published as *"Point-Based Tree Representation: A new Approach for Large Hierarchies"* at the IEEE Pacific Visualization Symposium 2009 and been extended to trees with spatial constraints in *"Point-Based Visualization for Large Hierarchies"* which is accepted with minor revisions for the IEEE Transactions on Visualization and Computer Graphics.

As for different graph types, the layered graph layout is identified as a design principle, which is particularly well suited to display bipartite graphs. This is concretely exploited by a visualization design that employs two layers of tabular node lists with edges in between them. Using this particular setup does not only express the very nature of bipartite graphs, but also allows to add further enhancements and interaction mechanisms that were originally devised for tables to this graph layout. New exploration techniques such as a script-based selection for automating complex interactions are introduced to ease the interaction with large tables. To underline the thereby gained versatility, the visualization technique is applied to three different use case scenarios from the life sciences. This work has been published as *"Visual Analysis of Bipartite Biological Networks"* at the Eurographics Workshop on Visual Computing for Biomedicine 2008 and its extension to time-varying model structures is submitted to Theoretical Computer Science under the title *"Constructing and Visualizing Reaction Networks from Pi-Calculus Models"*.

The third and last part of the discussion finally details the necessary considerations on the *task level*, which influence the design of explorative graph visualizations. As a first step, this concerns the decision whether to perform an exploration task visually or computationally, and how to sensibly combine these individual visual and computational analysis steps to a single exploration workflow. This touches upon the mentioned issue of the representation's interactivity, as computational analysis steps are usually longer-running and thus preventing an interactive back and forth between their parametrization and exploration of their results. A combined workflow is given in the form of a visual analytics framework, which uses preprocessed metadata to streamline subsequent analysis steps. The resulting framework was published as *"A Framework for Visual Data Mining of Structures"* in the Proceedings of the 29th Australasian Computer Science Conference 2006.

As the next step, the necessity of supporting confirmative analysis alongside the explorative analysis is discussed. Without an indication of how significant a discovered pattern actually is, the value of such a finding is hard to judge. It remains unclear whether these are noteworthy occurrences that cannot be explained by pure chance, or not. Thus, graph visualizations should try to combine both notions of graph analysis, as it can be observed in recent visual analytics tools, too. The benefits of such a combination are shown for a use case taken from social network analysis, which has appeared as *"HoneyComb: Visual Analysis of Large Scale Social Networks"* in the Proceedings of the 12th IFIP TC13

Conference on Human-Computer Interaction 2009.

A last step discusses a task-centric visualization design approach for multiple data sets. This design approach is developed in response to a problem that explorative graph visualization shares with many other methods of analysis: the selection of appropriate data sources from a heterogeneous data pool for a given task. This challenge has already influenced the design of the aforementioned table-based visualization for hypergraphs which incorporates additional information from different online sources. But to approach this challenge in principal, it needs a more general understanding of the dependencies between data sets, views, and task, which can be gained through comprehensive modeling of these three levels. In this model, dependencies between data sources, multiple views, and the different steps of an analysis workflow are described using directed graphs. Given a certain task, an automated pruning of the modeled dependency graph yields valuable information about appropriate data sources and the order they will be used in. Such a model is given for a concrete biomedical use case, which has been developed in collaboration with the Caleydo biomedical visualization project team from the Institute for Computer Graphics and Vision of the TU Graz. Preliminary work on this approach appeared under the title *"Towards Multi-User Multi-Level Interaction"* in the Proceedings of the Workshop on Collaborative Visualization on Interactive Surfaces in 2009.

As with each step of the discussion another level is brought into focus and added onto the considerations already made, discussion and solution approaches broaden with each part – starting from the in-depth discussion of implicit tree representations, through the visualization solutions for certain data specifics and graph types, which exhibit a flexibility that leads to a wide spectrum of layout facets and adaptations, e.g., for geospatial and temporal data, all the way to the generic visual analytics frameworks for graph exploration and the task-driven visualization design approach that goes even beyond graph data. Yet, the common ground most of the presented approaches build upon is their common use for life science applications, as they arise in the interdisciplinary graduate school "dIEM oSiRiS", within which this thesis was conceived. An overview publication on the research conducted in the graduate school, which places some of the visualization techniques introduced in the following in this application context, was published as *"Regenerative Systems – Challenges and Opportunities for Modeling, Simulation, and Visualization"* in the Proceedings of the 4th ICST International Conference on Performance Evaluation Methodologies and Tools in 2009.

1.3 Structure of this Thesis

The structure of this thesis follows the argumentational path as previously outlined. Hence, the following Chapter 2 contains basic considerations concerning the graph types, representation paradigms, and tasks. The inherent problems and challenges of exploratory graph analysis raised in Chapter 1.1 are discussed in detail in Chapter 3. It is then followed by the three mentioned parts of discussing solution approaches and concrete use cases on the level of:

- **Representation:** Chapter 4 introduces the design space of implicit hierarchy visualizations, its implementation, as well as examples of novel visualization designs generated by using it.
- **Data:** Chapter 5 presents the point-based visualization as a space-efficient representation for large tree, as well as the table-based visualization for bipartite graphs.
- **Task:** Chapter 6 discusses the visual analytics framework for graph analysis, the combination of exploration and confirmation, as well as the task-centric visualization design for determining suitable data sets to perform a given task.

A short summary of the discussion and worthwhile directions for future work are given in the end in Chapter 7.

Chapter 2

Basic Considerations

This chapter summarizes and reviews the terms and concepts associated with explorative graph visualization. Following along the lines of the three levels explorative graph visualization touches upon, this chapter treats in three parts the underlying data, their graphical representation, and possible exploration tasks. In the end, a fourth part brings them together by discussing their combination – the applicability of the presented representations to the different graph classes and their use for the different tasks.

2.1 The Underlying Data: Graphs

Graphs are a mathematical concept that serves as an abstraction for many real world occurrences of connectedness or relation. In its basic form, it captures four aspects of the notion of connectedness:

- the entities that are connected as *vertices* or *nodes*,
- the connections between these entities as *edges* or *links*,
- the orientation of the connections in the from of *directions*,
- the strength of the connections as *weights*.

The study of graphs and the investigation of their properties is subject of the mathematical discipline of *graph theory*. It divides graphs into a number of *graph classes* and subclasses depending on the characteristics of the set of edges $E = \{e_1, e_2, \ldots\}$ connecting the nodes of the node set $V = \{v_1, v_2, \ldots\}$. These graph classes range from *independent sets*, having the smallest possible edge set $E = \varnothing$, all the way to *hypercliques*, having the largest possible edge set $E = \wp(V) \setminus \{\varnothing\}$ with $\wp(V)$ denoting the power set of V. As of April 2008, the Information System on Graph Class Inclusions[1] lists more than 1,000 different graph classes, about 200 of which are thoroughly surveyed in [BLS99].

The most essential graph classes in the context of this thesis are: *hypergraphs* H, *networks* N, *bipartite graphs* B, and *trees* T. They are intrinsically related by inclusion

[1] http://wwwteo.informatik.uni-rostock.de/isgci/

as $H \supset N \supset B \supset T$. This implies for example, that each tree can also be thought of as an instance of a bipartite graph and be modeled as such. For the scope of this thesis, a number of shared characteristics are defined for all four graph classes:

- The **set of vertices** V is defined as a finite set of atomic entities. This explicitly excludes so-called *compound graphs* in which vertices can again be graphs on their own.
- The **set of edges** E forbids multiple edges between the same vertices, unless being of different orientation. As the set of vertices V is finite, the edge set E must also be finite. *Self-edges* connecting vertices with themselves are allowed.
- All edges $e \in E$ are **directed**, $e = (V_{\text{from}}, V_{\text{to}})$ with $V_{\{\text{from,to}\}} \in \wp(V) \setminus \{\varnothing\}$. This implies $(V_{\text{from}}, V_{\text{to}}) \neq (V_{\text{to}}, V_{\text{from}})$, which is why both edges can simultaneously exist in E, unless e is a self-edge with $V_{\text{from}} = V_{\text{to}}$.
- All edges are **weighted** by a weight function, $\forall e \in E : f_w(e) \mapsto \mathbb{R}$. For unweighted graphs, $\forall e \in E : f_w(e) = 1$ is assumed. Additionally, for all missing edges that could theoretically exist but are not present in a graph, the weight function is defined as $\forall e \notin E : f_w(e) = 0$.

Hypergraphs form the most general class of graphs presented here. Their edges E, also called *hyperedges*, can connect any arbitrary number of nodes $V_{\text{from}} \in \wp(V) \setminus \{\varnothing\}$ with any other number of nodes $V_{\text{to}} \in \wp(V) \setminus \{\varnothing\}$. V_{from} and V_{to} do not need to be disjoint (self-hyperedge). A fundamental treatment of hypergraphs is given in [Ber89], and an overview specifically for directed hypergraphs can be found in [GLPN93]. Many real world examples can be abstracted as hypergraphs [GS98]. A prominent example are biochemical reaction networks [KHT09]. Other applications are the representation of (geo-)spatial relations [ZG02] and the interference modeling in mobile communication systems [SS98].

Networks are the graph class most commonly associated with the term “graphs". They capture the intuitive notion of connections as a binary relation by only allowing edges between two nodes $|V_{\text{from}}| = |V_{\text{to}}| = 1$. Since both sets V_{from} and V_{to} have only one single element, the set notation is dropped altogether for this graph class, as well as for both of the following subclasses. Hence, an edge e is defined as $e \in V \times V$ with $e = (v_{\text{from}}, v_{\text{to}})$. *Social networks* [WF94] are a prominent example among numerous other real world examples being modeled as networks [NBW06].

Bipartite graphs differ from networks as their set of nodes V can be partitioned in two disjoint node sets V_1 and V_2 so that $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$. This means edges run only in between the two node sets V_1 and V_2, which are by themselves independent sets. This restriction comes naturally, when modeling connections between two different types of entities – e.g., business networks connecting directors and boards [RA04], or consumer-services networks connecting readers and books [LGC09]. A thorough treatment of bipartite graphs is given in [ADH98].

Trees are the most restricted class of graphs in this list. As the main difference to networks and bipartite graphs, they are *connected* and *acyclic*. This means for any two

given nodes $\{a,b\} \subseteq V$ there exists a unique *path* between them. Such a path is a finite series of consecutive edges $e_1,\ldots,e_n$ with $e_i = (v_{i-1}, v_i)$ and $e_{i+1} = (v_i, V_{i+1})$ having a node v_i in common and $e_1 = (a, v_1), e_n = (v_{n-1}, b)$. In this definition, it is the existence of such a path for all pairs of nodes that ensures connectedness, and it is the uniqueness of any such path that guarantees acyclicity. In analogy to the term "tree", *leaves* denote nodes that have only one edge connecting it to the rest of the tree. In many cases, one of the nodes is defined as the *root*. This node $r \in V$ serves as the center of the tree from which it "grows outwards". The root can be any vertex, even a leaf, and is defined by the real world semantics behind the tree. For example, in case of a managerial hierarchy of a company the natural root would be the CEO. Trees with a defined root are called *rooted trees* $T_R = (V, E, r)$. All trees considered in this thesis are rooted. In rooted trees, an implicit directionality is given by orientating all edges away from the root towards the tree's leaves. For the example with the managerial hierarchy, this results in all edges pointing from the manager to the managed. Apart from this implied directionality, *directed trees* do not play a major role and are not considered in this thesis. In computer science, there are multiple works investigating trees from different points of view, e.g., algorithmics [KE00], database storage [Cel04], optimization [WC04], and data analysis [BG91]. Their application fields include biology using phylogenetic trees and genealogy employing family trees.

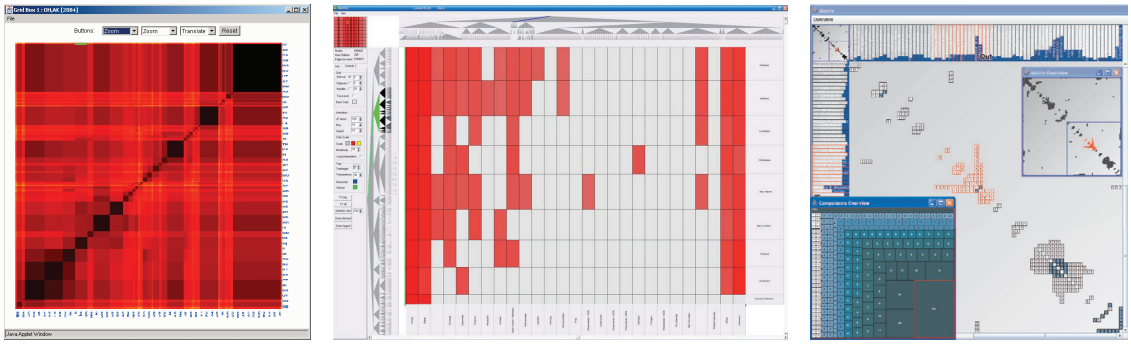
2.2 The Graphical Representation: Graph Visualization

There are three major types of graph representations that are used for most of the existing graph visualizations: *matrix representations*, *node-link representations*, and *implicit representations*. Exceptions that cannot be placed in any of the three are either hybrid representations combining these representational paradigms, or abstract representations as they are described in [Spi03] and which are mostly of interest for the combinatorial problems they pose. The common types of representations are conceptually introduced in the following, a more technical overview can be found in [vH08].

2.2.1 Matrix Representations

Matrix representations are a direct graphical interpretation of a graph's *adjacency matrix* M_{adj}. This is a quadratic $|V| \times |V|$ matrix with each row and column representing one node $v \in V$ and each matrix cell containing the corresponding edge weight $M_{adj}(i,j) = f_w(e)$ iff $e = (V_{\text{from}}, V_{\text{to}})$ and $v_i \in V_{\text{from}}, v_j \in V_{\text{to}}$. Note: in hypergraphs, a pair of nodes can be part of multiple edges and thus the value of a matrix cell may be an aggregate of multiple edge weights. The graphical adaptation uses the matrix as a grid and encodes the edge weights usually by coloring the matrix cells according to a given color scale.

The challenge posed by this kind of representation is how to find a "good" ordering for the nodes and thus for the rows and columns. The goodness of such an ordering can be defined in many ways. This can be any ordering ranging from simple sorting of rows/columns according to a given node attribute to rather complex arrangements that bring out characteristic features of the graph. The latter is done by a permutation of the



(a) (b) (c)

Figure 2.1: Graph visualizations making use of the matrix representation: (a) Graph Sketches [AFK01], (b) MatrixZoom [AvH04], and (c) MatrixExplorer [HF06].

rows and columns according to their similarity, placing similar rows/columns close to each other using, e.g., clustering techniques.

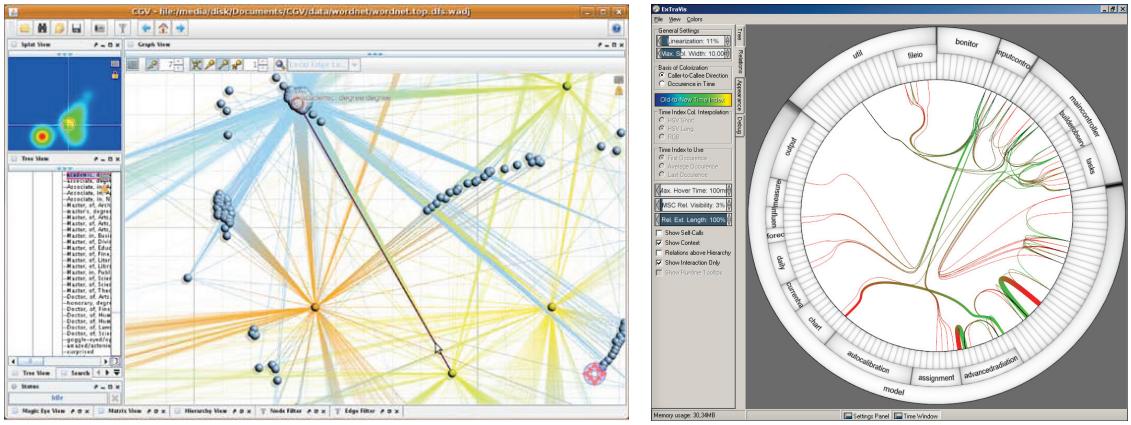
A comprehensive overview on established matrix visualizations for graphs is given in [Hen08] and [Sch04], some examples are shown in Figure 2.1. The main benefit of matrix representations is their focus on the graph's edges. It leaves them most of the available drawing space, whereas the nodes become mere indices for rows and columns and appear only as row and column labels at the side of the matrix. Its downside is its quadratic space requirement which allocates one row and column per node – one matrix cell for each edge possible.

2.2.2 Node-Link Representations

Node-link representations depict nodes as points or objects in 2- or 3-dimensional space and the edges between them as curves. This kind of representation is by far the most common one. It has its roots in circuit board design where logical gates and other electronic components must be positioned and connected in a space-efficient way – very much as nodes in a good graph representation. The classic work on node-link representations [BETT99] defines a list of drawing conventions (e.g., curved vs. orthogonal links), aesthetics (e.g., minimal edge crossings), and constraints (e.g., certain subgraphs laid out in a particular shape) that node-link layouts should seek to fulfill.

Yet, in general it is not possible to fulfill all of the different layout properties at once as desired layout constraints may contradict each other. And even when prioritizing them, most layout problems remain NP-complete [DPS02]. Hence, unless P = NP, its optimal solution can only be approximated in reasonable running time. Towards that end, the layout problem is formulated as an optimization problem. This is usually done using physical analogies [Bra01] like springs and electrical forces [Kob10]. Once formulated, it can be solved using standard optimization methods like steepest descent [Tun99], genetic algorithms [JwWf03], or artificial neural networks [CS96].

Apart from the actual layout, a wealth of options are known that postprocess a once determined layout to improve its appearance and readability. Examples are:



(a) (b)

Figure 2.2: Graph visualizations making use of the node-link representation: (a) Coordinated Graph Visualization [TAS09] and (b) ExTraVis showing its Circular Bundle View [CHZ+07].

- **bundling of edges** by combining links going in the same direction [HvW09a]
- **removing node overlap** by repositioning nodes intersecting one another [DMS05, DMS06]
- **labeling** by unobtrusively placing labels on the layout [DKMT07]

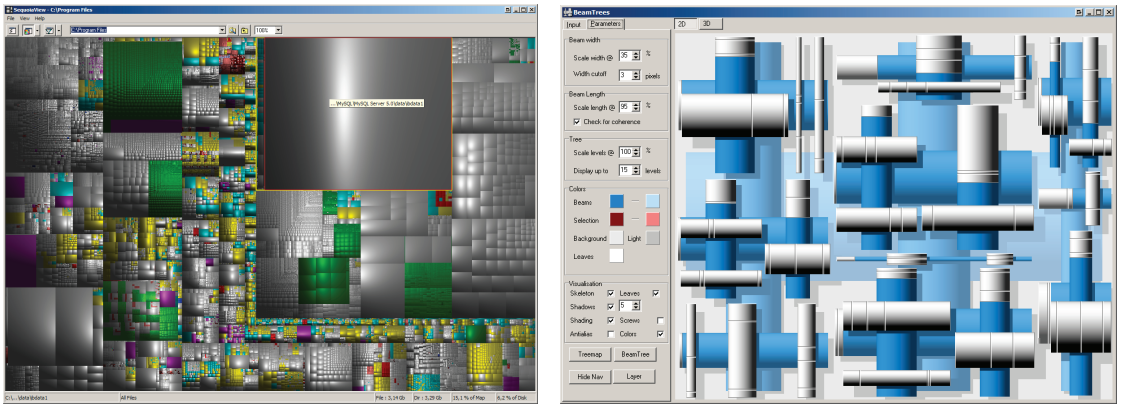
In general, this representation is well suited to display vertices as well as edges in a balanced way that gives equal room to both of them. Another positive aspect is their widespread use and recognition even by non-technical audiences. Yet, the computational cost of solving the layout problem remains high, as approximation algorithms have usually a polynomial runtime complexity and thus need their time to produce high quality layouts. The given examples of postprocessing already indicate merely solving the optimization problem does not necessarily produce very orderly layouts and needs further refinement.

Two examples for visualizations utilizing node-link representations are shown Figure 2.2.

2.2.3 Implicit Representations

Implicit representations differ from node-link representations by their lack of explicitly drawn edges. Instead of using curves, edges are encoded into the relative position of the vertices. There exists a wealth of such representations that use, e.g., overlap, adjacency, inclusion, or visibility. These representations are often called *space-filling* as the absence of edges allows to place the objects representing the vertices as tightly as possible in the available drawing space.

It is this property which also implies the major layout challenge for this kind of representation: to find an arrangement of the nodes that represents a given graph according to a chosen positional relation and uses the drawing space effectively. Depending



(a) (b)

Figure 2.3: Graph visualizations making use of the implicit representation: (a) SequoiaView showing a Treemap [vWvdW99] and (b) 2-dimensional Beamtrees [vHvW02].

on the used layout strategy, this can be fast and simple (e.g., subdivision of the available space in linear time [Joh93]), or time-consuming and complex (e.g., packing of the available space being NP-complete [GJ79]). Additionally, there exists a set of desirable layout properties [Wat05], like an aspect ratio of ≈ 1 for the vertex objects/regions – not unlike the constraints given for aesthetic node-link layouts. Examples of implicit representations are abundant – e.g., *Treemaps* [JS91, Shn92, Joh93] (using inclusion), *Icicle Plots* [KL83] (using adjacency), *2D Beamtrees* [vHvW02] (using overlap), and visibility representations [BEF+98]. Two visualizations using implicit representations are depicted in Figure 2.3.

The upside of using implicit graph representations is their efficient usage of drawing space for the node primitives. This allows to emphasize node attributes as object size, to embed objects like images or labels in the nodes to be clearly visible [Bed01, TON04], or to alter surface properties like shading and texture [HVvW05] to add further information to them. Their drawback is that positional relations like overlap and inclusion naturally create visual clutter by overplotting parts of the representation, which can hamper perception if not carefully laid out. Additionally, the directionality of edges may get lost, depending on the chosen positional relation. For example, overlap can easily show directionality, as either node *A* overlaps node *B* or the other way around. But adjacency or visibility are symmetric relations and thus unable to encode any direction.

2.2.4 Hybrid Representations

These combinations of different representational paradigms are a fairly new development and still not very wide-spread. Their idea is to mix representations where it is possible in order to compensate their individual disadvantages. Yet, this also adds a new layout challenge to the ones of the individual representations – namely to determine for which part of a graph to use which representation. Some hybrid representations leave this completely to the user who determines via interaction which part to show in which way. Yet,

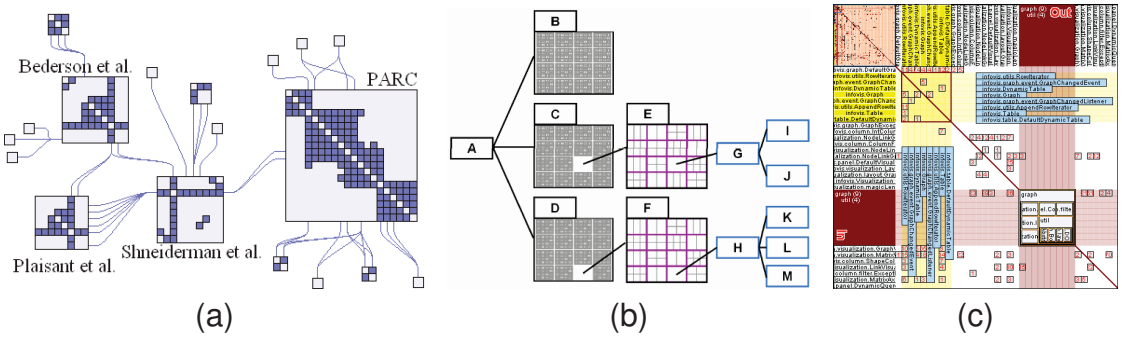


Figure 2.4: Graph visualizations combining different representations: (a) NodeTrix [HFM07], (b) Elastic Hierarchies [ZMC05], and (c) embedded Treemap in a matrix view [RMF09].

it is also possible to use automatic methods in a preprocess to determine this [AMA07]. Examples for possible combinations are:

- **matrix + node-link** – *NodeTrix* [HFM07]
- **implicit + node-link** – *Elastic Hierarchies* [ZMC05]
- **matrix + implicit** – embedded Treemaps [RMF09]

All three examples are shown in Figure 2.4. An attempt to combine all three approaches in one hybrid representation is so far not known – even though it seems rather straightforward to combine, e.g., NodeTrix with Elastic Hierarchies if the need arises.

While such hybrid representations have the potential of combining the positive aspects of the representations involved, their use also requires an expertise in each of the individual representations. Furthermore, as the different representations allow different interaction paradigms, devising a coherent interaction concept for such hybrid representations is a challenge by itself.

2.3 The Task: Exploration

So far, no authoritative list of analysis tasks for graphs has been established. Yet, the publications on this matter mostly agree on their classification according to the *analysis target*, meaning the part of the data the tasks are performed on. Thus, from a high level perspective, one can differentiate between *attribute-based tasks* (ABT) and *topology-based tasks* (TBT). Their difference lies in the fact that attribute-based tasks can be performed on either node or edge set, whereas topology-based tasks need to consider both sets in conjunction. Hence, attribute-based tasks are completely independent of the actual graph structure and can be perceived as ordinary analysis tasks on multivariate data sets. Typical instances including examples for both task categories are:

localize – means to find a single or multiple nodes/edges that fulfill a given property

- ABT: Find the edge(s) with the maximum edge weight.
- TBT: Find all adjacent nodes of a given node.

quantify – means to count or estimate a numerical property of the graph or parts thereof

- ABT: Give the number of all nodes.
- TBT: Give the indegree (the number of incoming edges) of a node.

sort/order – means to enumerate the nodes/edges according to a given criterion

- ABT: Sort all edges according to their weight.
- TBT: Traverse the graph starting from a given node.

An overview of various attribute- and topology-based tasks can be found in [Lee06, ch.6] and [LPP+06], which gather a number of evaluation tasks from the InfoVis contest in 2003, as well as from [AES05]. It should be noted that these tasks are not necessarily atomic but still rather basic and fundamental. Real-world tasks are often more complex combinations of such fundamental attribute-based and topology-based tasks. An example for such a combined task would be: "find the proportion of links from a node that go to each category for every node" [SA06].

2.4 The Combination: Explorative Graph Visualization

After defining the underlying data (graphs), their representation (visual encoding), and their exploration (high-level tasks and actions), this part finally brings all of it together. It discusses which representation is suitable for which kind of graphs, and what are the tasks that can be accomplished with the help of these representations.

2.4.1 Combining Data and Representation

The applicability and use of the different representational paradigms is by far not the same for all graph classes. While some representations can be considered as standard for a given graph class, others may only exist as experimental visualization techniques and are not generally applied. In the following, the four introduced graph classes will be examined with respect to their possible representations.

Hypergraphs

Representations for hypergraphs are always node-link representations. There exist two different styles of them: the *edge standard* and the *subset standard* [Mäk90], both shown in Figure 2.5. They differ in how the links are drawn – either with open curves (edge standard) or closed curves (subset standard). The latter can only be used for undirected hypergraphs, as it encircles the nodes connected by an edge with no possible differentiation between V_{from} and V_{to}. Most algorithms to automatically generate representations of both

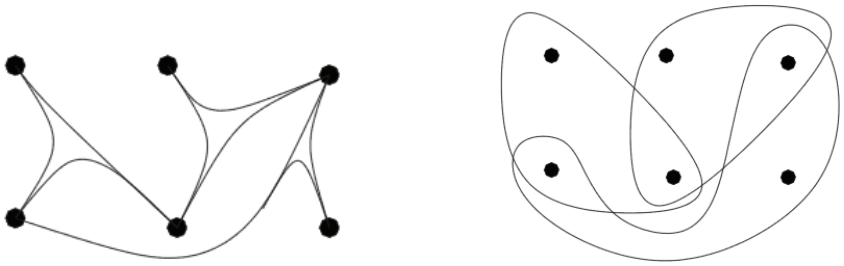


Figure 2.5: The two different node-link layout styles for hypergraphs: (a) edge standard and (b) subset standard [BE00].

kinds use a force-directed approach as it is often the case for node-link representations in general. An example for a realization of the edge standard is the Hypergraph-Draw system [KLMS95] and the subset standard is implemented in the PATATE system [BE00]. In general it can be said that node-link representations in the edge standard are most widely used, whereas the subset standard is less prevalent, as it does not scale well for large hypergraphs. And even for smaller hypergraphs it is problematic to achieve readable layouts with the subset standard.

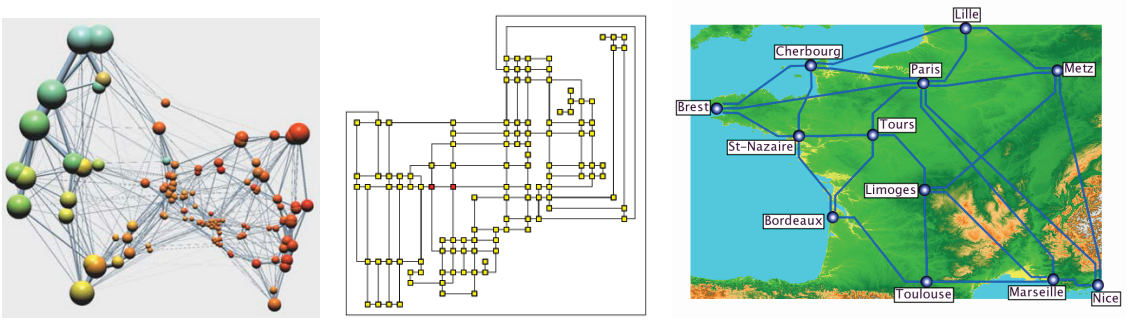
Implicit techniques and matrix representations are basically non-existent for hypergraphs. First thoughts on implicit representations (visibility representation, adjacency of triangles) are given in [dM99]. Yet, these are only possible to establish for the limited subclass of planar hypergraphs and thus not usable in general. For matrix representations, the problem is a different one, as there does not exist an agreed upon generalization of adjacency matrices for hypergraphs. An example is given in [vN79], which generalizes adjacency to the notion of reachability. Another disadvantage of adjacency matrix representations for hypergraphs is the lack of a one-to-one correspondence with the actual hypergraph, meaning that it is impossible to reconstruct the original hypergraph from its matrix representation. These issues have probably lead to the situation that there exists no matrix visualization for hypergraphs so far.

Networks

Networks are mostly represented using node-link representations. In their case, different degrees of freedom in the placement of nodes lead to different layout types. In general, three layout classes can be identified[2]:

free – the nodes can be placed without being restricted in any way. This layout type is used when the placement of the nodes themselves should already show features of the network, e.g., clusters in the form of overcrowded regions and other parts of the network which are less dense.

[2]appeared as "Visualizing Graphs – A Generalized View" in Proceedings of the 10th International Conference Information Visualisation 2006



(a) (b) (c)

Figure 2.6: Examples for the node-link layout classes: (a) free, force-directed layout [vHvW04], (b) styled, grid-based layout [BW98], and (c) fixed, map-based layout [Rod05].

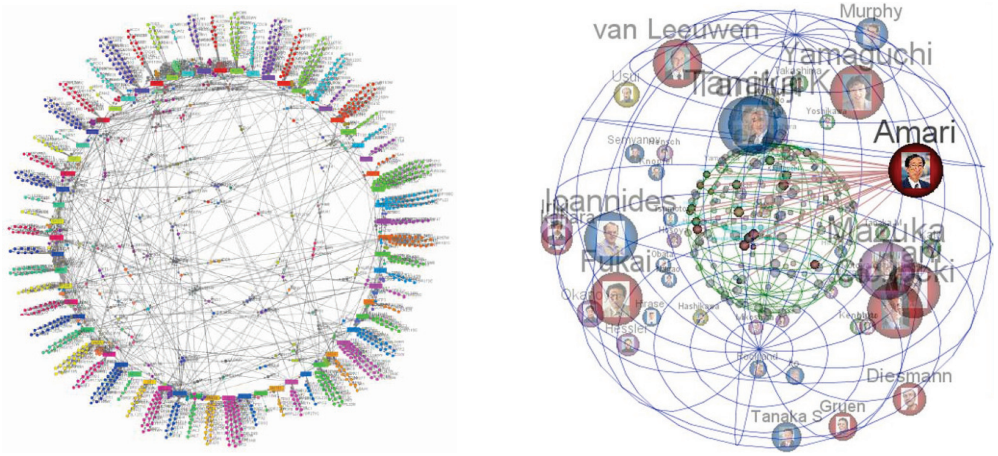
styled – the nodes are placed according to some layout style like the *Connected Ring Pattern* [GK06, DLR09, KSB+09] or the *Linear Layout* [Cim06]. This layout type can be used to impose an ordering on the nodes, to ensure an even spatial distribution of nodes, and to visually separate nodes from edges.

fixed – the node positions are predetermined and cannot be changed. Fixed layouts have a strong focus on computing well-routed edges, as this is the only part of the layout that can still be adjusted. An example is the edge rerouting algorithm given in [DK08].

Among these three layout types, the free layout is the one most widely used as it is also the most flexible. Styled layouts are quite common in the visualization of data networks and network security, probably because in this area, network nodes are always arranged in some kind of network topology anyways – ring, grid, star, etc. Fixed layouts are only used in special cases, e.g., for transportation networks where the nodes are associated with geolocations or for visualizing co-occurrences in documents [Wat02]. Examples for all three node-link layout classes are given in Figure 2.6.

Apart from node-link visualizations, matrix visualizations are also applicable to networks. Yet, compared to the wealth of node-link representations, their number is diminishingly low. Matrix representations make most sense for very dense graphs, meaning graphs that have many edges. Otherwise a lot of the drawing area will be wasted for empty matrix cells. As node-link representations tend to be only poorly readable for dense networks because of a lot of clutter from the edges, hybrid techniques like NodeTrix (see Chapter 2.2.4) suggest to combine both. This way, dense subnetworks are displayed using matrices and sparser parts of the network are displayed using nodes and links.

Implicit representations are not used for the visualization of networks, even though some examples for certain subclasses of networks have been proposed. For example, there exist Treemap variants that generate implicit representations for directed acyclic graphs, a special subclass of networks [TTT07, KMBG07].



(a) (b)

Figure 2.7: Examples for the bipartite layout styles: (a) a circular style being use for one node set (anchors), the other node set can be placed freely [Mis06] and (b) a spherical style that fixes one node set to the outer sphere and the other one to the inner sphere [NUUT07].

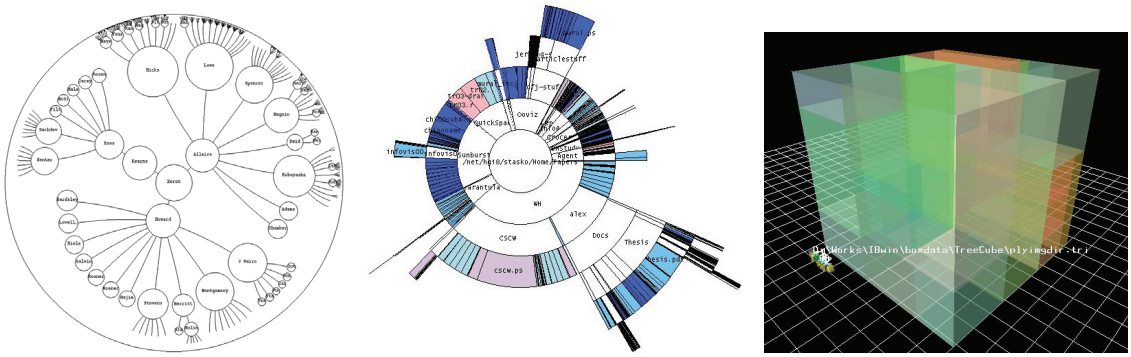
Bipartite Graphs

Bipartite graphs are not much in the focus of visualization at all. But the few visualizations that have been specifically designed for bipartite graphs agree all in one point, namely that it is important to differentiate between the nodes of both node sets. The node-link approaches use styled layouts and/or color-coding to achieve this. The used layout styles are mostly circular/spherical and can be applied to one of the node sets as in *Anchored Maps* [Mis06, IMT09] or both node sets [NUUT07, GGL08]. Both variants are exemplified in Figure 2.7. Color-coding is usually used in conjunction with a styled layout. Yet, if the drawing conventions forbid a styled layout as it is the case for certain application fields like biology, the color-coding may be the only distinction between different node types. A color-coded example for a set of interconnected metabolic pathways is given in [BCL+07].

Node-link diagrams are certainly the most important representations for bipartite graphs. Implicit representations are rarely even considered, as they only allow to represent a subset of all possible bipartite graphs [DH94]. Matrix representations for bipartite graphs differ somewhat from usual matrix representations, as they do not use a quadratic adjacency matrix but a rectangular one with one node set as columns and the other one as rows. So, if one node set is much larger than the other this can lead to an extremely unfavorable aspect ratio for the representation.

Trees

General trees are mostly represented with a node-link layout. Yet, the majority of representation techniques for trees specifically concern rooted trees, as they occur in a wide variety of application fields. For rooted trees, implicit representations are as popular as



(a) (b) (c)

Figure 2.8: Tree visualization examples: (a) radial, node-link – *Hyperbolic Browser* [LR96], (b) radial, implicit – *Sunburst* [SZ00], and (c) axis-parallel, implicit – *Treecube* [TON03].

node-link layouts. Both appear in a variety of styles, the most prominent ones being *radial* and *axis-parallel* layouts. The radial style puts the root node in the center of the layout and subdivides and arranges subsequent hierarchy levels on concentric circles around the root. This has the advantage that the available space increases with each level, as does the overall width of the tree. The axis-parallel layouts on the other hand layer the individual levels horizontally and vertically. Examples for both are given in Figure 2.8.

The node-link layout of rooted trees is computationally much easier than for general networks, as the layout does not need to approximate some physical model of the nodes and edges. Instead, specifically adapted layout algorithms are available – e.g., the Walker algorithm [Wal90, BJL02] running in linear time. The same goes for implicit layouts of trees which can be generated efficiently through hierarchical subdivision in 1D (e.g., Icicle Plots [KL83]), 2D (e.g., Treemaps [JS91, Shn92, Joh93]), or 3D (e.g., TreeCube [TON03]).

Matrix visualizations are rather uncommon for trees, as trees are very sparse, having only $|V|-1$ edges. Using only this few matrix cells in a $|V| \times |V|$ matrix would leave a lot of space unused. Therefore, it makes sense to detect tree structure in subgraphs of a larger network and substitute their region in the overall matrix with a space-filling technique like a Treemap, as it is proposed in [RMF09].

2.4.2 Combining Representation and Task

Whether a task can be carried out well or not with a given representation depends on a number of influence factors, like the data size and the interaction facilities provided by the representation. These usability aspects are qualitatively examined in several user studies, comparing different implementations of the presented representational paradigms. While their results are difficult to generalize and compare beyond their respective individual setups [ZK08], there are a number of obvious observations that can still be made – e.g., the impossibility of pursuing edge-related tasks with implicit representations. The following

parts will briefly discuss these and give exemplary pointers to individually published user studies for further reference.

Matrix Representations

Because of their focus on edges, adjacency matrices are first and foremost great tools to carry out tasks on the edge set. As all edges are visible, individual edges can be localized either by their incident nodes through a simple look-up in the index sets of rows and columns, or by attribute value if it is color-coded in the matrix cells. The quantification of edges is easily done as the relative number of all edges being present vs. all edges being theoretically possible can be estimated from the matrix representation. By sorting rows and columns accordingly, individual edges can be grouped visually into denser clusters. The drawback of matrix representations lies in the fact that they can hardly aid in topology-based tasks if these exceed the local neighborhood. For instance, the locally confined task of determining all adjacent nodes of a given node translates into a simple scanning of a single matrix row. Yet for following a path through the graph, matrices are not the representation of choice. Path traversal is usually associated with node-link representations, which tend to perform better than matrices in this case [GFC05, KEC06]. As a result, solutions have been proposed that aid in perception of paths in matrix representations by overlaid links [SM07, HF07], as exemplified in Figure 2.9.

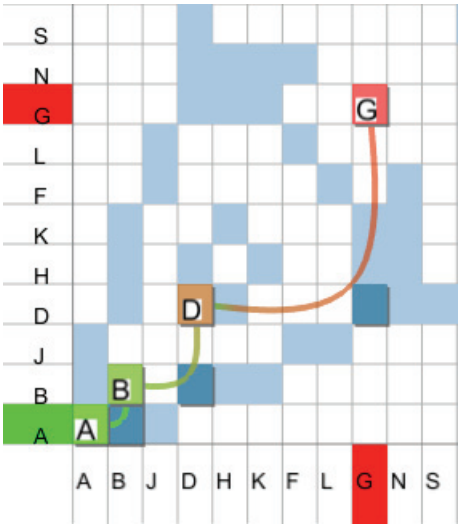


Figure 2.9: Overlaid path [SM07].

Node-Link Representations

As node-link representations show both, nodes and links, they are a good fit for topology-based tasks in general. Yet, it has to be noted that this assumes a suitable layout [MBK97, PCA02, HHE05] and graphical mapping [HZBK08, HvW09b], as both have been shown to influence task performance for node-link representations considerably. A natural example for a layout-dependency occurs for the task of sorting nodes according to a given attribute. This is only feasible in combination with a styled node-link layout (see Chapter 2.4.1) that allows to position the nodes in an apparent order. As node-link layouts tend to suffer from visual clutter with increased size (large node set) and density (large edge set), additional mechanisms may be needed to pursue certain tasks. As an example, in a large graph, neighboring nodes may be located in some distance from each other, depending on the layout. In order to still be able to quickly determine all neighbors of a given node, mechanisms like the *Bring Neighbor Lens* [TAvHS06] illustrated in Figure 2.10 have been developed.

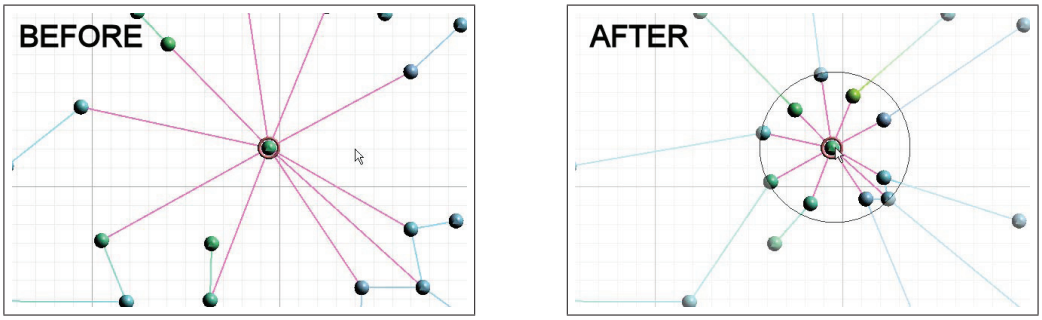


Figure 2.10: The Bring Neighbor Lens [TAvHS06] pulls-in adjacent nodes.

Implicit Representations

The implicit representations exclusively cover the node set of a graph. They are usually perceived as being very attribute-centric and thus good candidates to accomplish attribute-based tasks. At least for the specific case of Treemaps, this has been confirmed for tasks involving judgement of similarity and homogeneity [WTM06] and a quantification task on nodes [AK07]. Building on this advantage, especially techniques using the inclusion relation have been further adapted to better aid in comparison [BHvW00] and ordering tasks [SW01]. Techniques using adjacency instead of inclusion still show node attributes very well. Yet, at the cost of more drawing space they also emphasize on the structure. For certain techniques this has been reported to have a positive effect on both attribute-based as well as topology-based tasks [SCGM00, BN01]. At least for topology-based tasks, this appears to hold true also in case of 3-dimensional variations of adjacency representations: a user study on *Steptrees* (see Figure 2.11(a)) suggests a better performance for such tasks when compared to Treemaps [BCS04], [Bla06, ch.2.4]. Overlap techniques like the *Cascaded Treemap*, shown in Figure 2.11(b), try to compromise between space usage on one side and visualizing attributes as well as structure on the other side. Yet, at least for 2D Beamtrees, an overlap technique shown in Figure 2.3b, this compromise seems to come with a price: a user study found users to perform slower and less accurate for the attribute-based task of locating the three largest nodes [vHvW02], [vH08, ch.5.3].

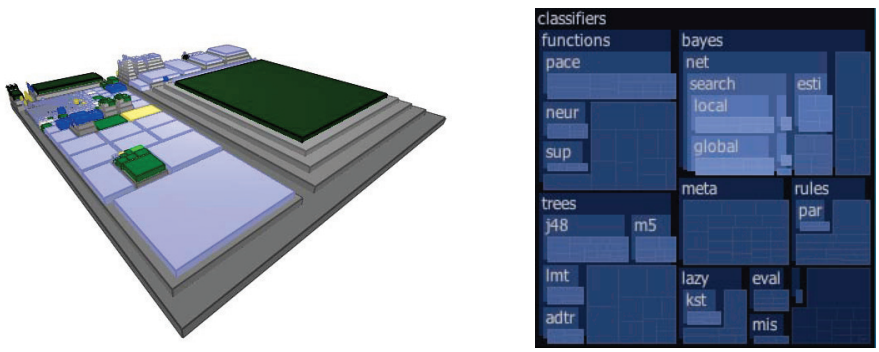


Figure 2.11: (a) Steptree [BCS04, Bla06], (b) Cascaded Treemap [LF08].

2.5 Summary

This chapter outlined and related the concepts of graphs, representations, and tasks in the notions as they appear in this thesis. The different graph classes (hypergraphs, networks, bipartite graphs, trees) have been recognized as having different structural characteristics, which makes each of them applicable to modeling real-life scenarios that match well with their respective structure. Their visualizations fall under one of three representational paradigms (matrices, node-link diagrams, implicit visualization). Each of which exhibits different benefits and drawbacks for displaying a certain graph class. Additionally, each representation is more or less suitable to support both types of analysis tasks (attribute-based, topology-based). As data and task define a suitable representation, the described dependencies capture the necessary factors to derive tailored representations for different data/task-combinations. The inherent problems of doing so, as well as possible approaches to alleviate them are briefly discussed in the following chapter.

Chapter 3

Problem Discussion

Challenges and problems are abundant in explorative visualization. They arise on and affect all levels of the exploration: data, representation, and task. For the individual levels, this commonly includes:

- **on data level:** e.g., aspects of data quality (accuracy, precision), accessibility (formatting, privacy issues), and confidence (trustworthiness of source)
- **on representation level:** e.g., aspects of representation quality (readability, comparability), efficiency (screen utilization, rendering time), and interactivity (intuitiveness, consistency)
- **on task level:** e.g., aspects of task specificity (to user skill or user knowledge), decomposability (into easier subtasks or parallel subtasks), and clarity (consistent and concise description)

In the following, challenges arising specifically in graph visualization and graph exploration are discussed and the solution approaches offered in this thesis are related to them.

3.1 Challenges in Explorative Graph Visualization

From a representation perspective, there are in particular two influencing factors on the visualization: the characteristics of the input data and the specific needs of the task at hand. While this is a well known fact, it directly leads up to the specificities on data and task level, that produce the characteristic representational problems of explorative graph visualization:

- **on data level,** these are first and foremost the specifics of having graphs as input. Here, different or unknown graph classes and large graph sizes are of importance.
- **on task level,** this is mainly the uncertain nature of exploration which leads to unspecific and/or often changing exploration tasks.

Different or unknown graph classes require different representations, as some representations are better suited for certain graph classes than others. That is because their layout makes better use of the respective properties of a graph class while at the same time emphasizing these properties so that the very nature of a graph is visually expressed. At its core, this is a problem of mapping data (in this case graphs) to appropriate visualizations (in this case graph layouts). Yet, not always an optimal mapping is possible. A frequent example in exploratory graph visualization is the representation of an unfamiliar data set with its specific graph type and size not being known in beforehand. So, as both are yet to be determined in the course of the exploration itself, only a generic layout can be used without having that information already. Such a generic layout does indeed show the graph, but only in its most general nature. For example, in a network layout any bipartite or hierarchical notion will not be clearly conveyed. In the worst case, depending on the network layout, this leads to the situation that the actual graph type is completely hidden from the user. And this is not only a problem of representation, but also of the exploration process utilizing it: an unclear, convoluted layout that already obscures the obvious characteristic traits of a graph is even less capable to communicate more subtle, underlying facets of the graph data and to interact on it. A problematic mapping from graph data to a representation becomes even more severe when something is shown or at least visually hinted at, which is not part of the data. This can easily occur, for example, if a node-link layout does not adhere to the usual drawing aesthetics. If the user expects these aesthetics to be met, e.g., a minimum number of edge crossings, and is presented with a poor and unnecessary complex layout instead, it is easy to falsely deduce a very dense and highly interwoven graph structure from it.

Large graph sizes are an immanent problem for sophisticated layout algorithms which usually have a polynomial runtime complexity, as well as for the exploration process, as an overcrowded, visually cluttered layout is more hindering than helping the exploration along. Yet, finding the right balance between a good (enough) layout and an acceptable computation time is far from trivial. This is because layout quality and computation time depend not only on the size and complexity of the input graph, but also on the available resources like display space. If one of these influencing factors changes significantly, layout quality and the layout time needed to achieve it must be rebalanced again. In addition to this, not only the representation but also the exploration of large graphs is challenging. Here, the two factors of interaction cost measured (e.g., in completion time or number of mouse clicks [Lam08]) and accuracy/reliability of thereby gained analysis results play an important role and are commonly chosen in user studies to evaluate representations with respect to given tasks. Finding a good trade-off between both is not only difficult but also highly task-dependent. Depending whether an exploration task emphasizes on speed, accuracy, or any weighted combination of both, this should be reflected and supported by the representation.

The uncertain nature of exploration leads to a vague description of the actual task to perform. This is far from uncommon, as only with the first visual overview of the data a user may get an idea of what aspect might be worthwhile to investigate and how to go about it. And even if a concrete task is given, it is also the very nature of exploration that a task often changes midways as new insights are discovered that lead the exploration in

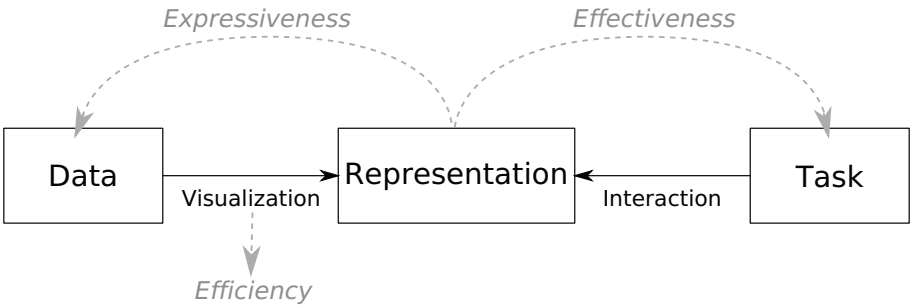


Figure 3.1: A schematized version of the principal visualization criteria.

a different direction for which the current representation may not be suitable.

All in all, these challenges often prevent to find a good match between the given graph, the exploration task at hand, and a suitable representation to show the former and support the latter. Yet, this is not surprising as it can be seen that each of the above mentioned challenges conflicts with one of the visualization quality criteria of expressiveness, efficiency[1], and effectiveness. In short, the three criteria stand for the following:

- **Expressiveness** means that the visualization truthfully shows all the data and only the data. This criteria is apparently afflicted by the issue of the different or unknown graph classes, as it is not ensured that the graph class is truthfully conveyed by the representation.

- **Efficiency** means that the visualization of the data is accomplished in reasonable quality with respect to the needed resources, such as time or screen real estate. Efficiency is especially hampered the larger the input graphs get, as finding a runtime efficient and screen efficient layout becomes harder.

- **Effectiveness** means that the visualization is useful to the user which also includes for pursuing the given exploration task. This aspect is particularly hard to achieve when considering the uncertain nature of exploration, as it is unclear towards which task to gear a representation.

How these concepts are interrelated is shown in Figure 3.1. Here, the black parts of the diagram show the underlying dependencies of the representation with respect to data and task. Whereas the gray parts are the quality criteria as stated above and applied to each of the individual parts of the schematic concept.

[1]Especially in visualization, the criteria of *efficiency* is often implied by the broader term of *adequacy* or *appropriateness*. Yet, as in graph visualization the notions of *algorithmic efficiency* and *space efficiency* play such an important role, the term *efficiency* is used throughout this thesis to emphasize this particular aspect of *appropriateness*.

3.2 Established Solutions in Explorative Graph Visualization

Representational problems can only be solved on the representation level up to a certain degree or severity. For example, if the number of nodes in a graph exceeds the number of available pixels for its representation, even the best representation must fail. Beyond that, synergistic solutions combining changes on multiple levels are needed. In the following, selected examples for each of the problem classes sketched previously are given and established solutions for them are discussed. This includes purely representational solutions and more comprehensive solution approaches affecting data and task level as well.

Inappropriate mappings from graphs to representations occur for example when visualizing *tree-like networks*. These are graphs which are trees at their core but have a few additional, so-called *cross edges* that create cycles in the graph and thus make it a network according to the mathematical definition. As acyclicity checks run in linear time, it can be determined efficiently whether a graph is a tree or network, and an appropriate visualization can be automatically chosen. Yet, this approach has its shortcomings: a single additional cross edge on a tree will then completely change the graphical representation from a tree visualization to a network visualization – even though the underlying tree structure is still there and should be conveyed by an expressive representation. Up to a certain number of cross edges, this problem can be solved purely on the representational level by simply drawing the cross edges on top of the tree representation as proposed in [Voi01, FWD+03]. Yet, when the number of cross edges becomes too large and they threaten to completely occlude the underlying tree, a more comprehensive solution is needed. An example for such a solution is edge bundling as introduced in [Hol06]. It uses the underlying tree to (pre-)compute for the two incident nodes of each cross edge their least common ancestor within the tree. These ancestor nodes are then used in the subsequent layout process as control points for the edges drawn as splines – effectively "pulling them together" into bundles which occlude the underlying tree representation less. This exemplifies how even a few and simple to compute extra information on data level can be exploited by the visualization to achieve an expressive representation that would not have been possible otherwise.

Computing fast and space efficient layouts has been in the focus of graph drawing and graph visualization research from the beginning. Hence, a large number of specifically adapted layout algorithms are available to choose from. These range from speed-ups through GPU-acceleration [FT07, GHGH08] to highly space-optimized tree layouts [NH03]. Yet, they all have their limits with respect to the input size for which they perform well. Beyond that, additional measures must be taken on data level in order to keep the representation efficient. A common technique used in these cases is the hierarchical partitioning of the graph in a pre-computing step. Such a partitioning on the data level can then be used to either focus the representation on a smaller part of the given graph or to visualize the whole graph on a higher level of abstraction with fewer nodes and edges. This approach is used, e.g., in the Coordinated Graph Visualization system [TAS09]. This system even utilizes the resulting partition hierarchy in adjoint views to further improve the interaction across partitions and abstraction levels at the same time.

The inherent fuzzyness of exploration is usually countered with a mix of multiple coordinated views that provide suitable views for a wide range of exploration tasks as well as a plug-in or scripting facility to add further representations. Together with general interaction techniques that allow a simple inspection of a given graph through panning and zooming operations, this is as far as this problem can be alleviated on the representational level alone. More effective representations with sophisticated and specialized interaction is only possible upon concretizing the exploration workflow on task level. The prime example for this is probably the substitution of the rather vague task of general exploration by *Shneiderman's Information Seeking Mantra*: "Overview first, zoom and filter, then details-on-demand" [Shn96]. This gives a more concrete exploration workflow, which is still high-level enough to allow a certain degree of freedom during interaction – yet, concrete enough to be specifically supported by representations.

This thesis focuses clearly on solutions on the representation level. Yet, it does so under consideration for problems on data level and task level that are of relevance for the representation. As outlined in the following, detailing the effect of these problems on the representation and presenting representational solution approaches to counter these effects is the main concern of this thesis.

3.3 Steps Towards Novel Solutions in Explorative Graph Visualization

As the discussion up to here has shown, the mere task of designing a graph visualization is not an easy one. Hence this thesis approaches explorative graph visualization in three steps:

1. By communicating the problems and possible solution approaches on representation level: In the light of the sheer number of possible combinations of available representational paradigms for graphs, their individual layouts, the multiple possible styles and orderings of these layouts, as well as the concrete graphical encoding of their elements – finding a configuration that is expressive, effective, and efficient is tedious and error-prone. Hence, before attempting to match suitable representations to data and task, one must be aware of the different visualization possibilities and their intricate interplay. Only then, design choices guided by intuition and a good sense of aesthetics can be replaced by informed and experience-driven design decisions. Hence as a first step towards that direction, a prototyping approach capturing the design space of implicit tree visualizations is proposed. This approach serves to raise awareness for the influence of the individual design decisions on the overall visualization in a learning-by-doing fashion. Only with this awareness, it is possible to carry out a directed visualization design. Chapter 4 details the design space, the dependencies between the individual design dimensions, as well as selected dependencies between design decisions, tree characteristics, and exploration task.

2. By approaching the data level problems in the representation: So, while the first step spans an entire design space for the example of implicit tree visualizations and

deduces design heuristics from it, the second step illustrates the case of highly specialized and optimized interactive visualization methods as they cannot be deduced by simply finding the right parameter combination in a general design space. It introduces and concretely realizes two such visualization designs taking on the mentioned challenges on data level: special graph classes and large graph sizes. Hence, the first is specifically considering the space efficient layout of large trees with several 100,000's of nodes. Graphs of these sizes are a challenge by themselves and only a handful of techniques exist that are capable of displaying trees this large. Yet, the actual design challenge taken on with this approach is not only to pack as many nodes as possible in the available drawing area, but at the same time to preserve expressiveness by communicating the overall structure of the tree. The second novel visualization design is specifically adapted to the visual exploration of hypergraphs and bipartite graphs – both being graph classes occurring in many application fields, yet rarely treated visually. Both of these highly specialized approaches are detailed and exemplified in Chapter 5.1.

3. By addressing the challenges on task level by an integrated framework of the exploration process: It remains to integrate the presented solutions on the representational level into the overall graph exploration process. This is done in the form of a process framework that divides the exploration in different phases and aligns them in a generic exploration workflow. This framework puts the previously mentioned visualization techniques into a broader context and emphasizes the ties of visual graph analysis to computational graph analysis. With the inclusion of computational analysis steps comes also the possibility of integrating confirmatory elements into the exploratory analysis, which is shown for a concrete example to be highly valuable for judging exploration results. The resulting many possible combinations of different computational as well as visual exploration and confirmation steps applied to the case of multiple data sets from a heterogeneous data pool lead to the situation that some form of guidance for analyst may be necessary, in order to make the right choices of which data/view to choose for a given task. Such a mechanism is devised in the end by utilizing a comprehensive model of all three levels in Chapter 6.

3.4 Summary

This chapter listed the three main representational problems of explorative graph visualization as they are imposed on a visualization by the specificities of graph data and the exploration task: various or unknown graph types and sizes on data level, as well as an variable exploration workflow. Solutions for them can be devised on a purely representational level up to a certain point. Beyond that, comprehensive solutions spanning also data and task level are needed. This thesis presents several solution approaches on the representation level: a general one, encompassing the entire design space of implicit tree visualizations in the following chapter, as well as highly optimized ones for very large trees and for the rarely considered graph type of bipartite graphs. These solution approaches are linked to the broader exploration workflow via a process framework also capturing analytical steps on data level as well as aspects of interaction on task level and interrelating them.

Chapter 4

Assessing the Design Space of Implicit Tree Visualizations

Having outlined the problem space of explorative graph visualization in the previous chapter, the importance of expressive, efficient, and effective visualizations for graph exploration is evident. While many visualization techniques are known today, they are already fully adjusted and fine tuned towards the concrete data and task they have been developed for. Picking and parametrizing an existing visualization so that it fulfills all three criteria for one's own data and task is a challenge by itself. In order to do that, one must not only be aware of the wealth of existing visualizations, but also know their different options and parameters. Setting the parameters of a visualization only slightly different may completely shift the focus of it, emphasizing and accentuating entirely different aspects of the data. This makes the search space for an expressive, efficient, and effective visualization not only very rich and diverse, but also very large. This chapter aims at communicating this problem and raising awareness for the many different parameters involved by systematically assessing the multitude of available visualization options for a concrete example.

The entirety of all such visualization options forms a visualization design space. The most prevalent strategy to define a design space is to extract common design principles from a set of existing visualization techniques. Once identified, these common principles can be used as axes to span the design space. Despite not being a space in the strict mathematical sense of a vector space, it serves as an established analogy in many fields. An extensive example is given by Bugajska's framework [Bug05]. This approach addresses mainly the need for a theoretical classification of existing techniques. Beyond that, visualization design spaces capture some of the tacit knowledge of visualization designers. Once made explicit, they can be used to access an otherwise diffuse and large set of visualization techniques:

- individual visualization techniques are defined by a concrete set of visualization parameter values and can thus be pinpointed to a distinct position within this space, whereas classes of visualizations are defined by intervals of visualization parameter values and can be understood as regions or subspaces within the overall design space,

- concrete techniques as well as subspaces can be mapped to problem/data domains [KK94], and design guidelines leading to expressive, effective, and aesthetically pleasing regions of the design space can be identified, as it is done in the works of Bertin [Ber81] and Tufte [Tuf90],

- the design process can be seen as "navigation" or "exploration" of this space – in a directed or undirected fashion, respectively – where each change of a visualization parameter equals a displacement along the corresponding axis of the design space.

Devising such a design space for the very heterogeneous and vast set of all graph visualization techniques may be difficult but not impossible. Yet, the heterogeneity would result in a large number of dimensions corresponding to the numerous design options and parameters. The resulting huge design space would be hardly helpful, as it would be as intricate as graph visualization design itself. Hence, for the purpose of communicating the design principles involved and their interplay, this chapter focuses on the subspace of implicit tree visualizations[1]. This class of graph visualizations has a rich history, stemming from *Euler-Venn-Diagrams*, which date back about three centuries [Bar69]. With their modern equivalents having evolved into dozens of individual visualization techniques, scaling well up to millions of nodes [FP02], being in widespread use in many software applications [Che06, ch.4.7], and often even protected by patents [AOPW01, Stu06] – this important class of graph visualizations is a prime example for a parameter-rich and diverse, yet still manageable subset of the overall graph visualization design space.

In a first part, this chapter establishes the implicit tree visualization design space by examining most of the existing implicit visualization techniques from more than 25 years of research. From them, common solution principles are extracted and discussed – mainly under consideration of their support for attribute-based and topology-based tasks. In the end, these principles effectively span the design space and can thus be used to characterize any implicit visualization technique, regardless if it already exists or not.

The second part gives a brief overview on the more practical sides of such a design space definition. It discusses issues of an actual software realization, making it possible to browse the design space and providing true *Rapid Visualization Prototyping* [KC96, LKG07] for implicit tree visualizations. This prototyping permits the necessary hands-on interaction with the design space for familiarization with possible design options, as well as evaluating and testing existing or novel representation approaches for given data and tasks.

With the set of implicit tree visualization techniques made explicit by the design space definition and accessible by the prototyping software, the third and last part exemplifies how the design of novel visualizations benefits from it. It introduces a number of new visualization designs by showing how to make use of the design space along the lines of three different visualization development approaches. Without the awareness of the different design options and their concrete realization for prototyping, this would not be possible.

[1] accepted with minor revisions as "The Design Space of Implicit Hierarchy Visualization: A Survey" for the IEEE Transactions on Visualization and Computer Graphics

4.1 Defining the Design Space

The design space will mainly be discussed along the lines of the two most prevalent implicit visualization techniques: Treemaps [JS91, Shn92, Joh93] and Icicle Plots [KL83]. This can be done without loss of generality, as most of the implicit visualization techniques relate to one, if not both of them. Implicit visualizations that do not relate to either of the two are extremely rare. Usually, they are more of theoretical interest and can only represent certain types of trees, as it is the case, i.e., for *unit rectangle-visibility representations* [DEMHP07]. Because of that, they are only of limited interest for the explorative visualization of actual data and hence not discussed in the following.

Treemaps stem from the *Euler-Venn-Diagrams*[2]. Treemaps represent the parent-child relationship between two nodes through nesting the child inside the parent. The first concrete ideas on how to do that (e.g., the Slice-and-Dice layout) were brought up by Johnson and Shneiderman in 1991. As early as 1993, Johnson outlined in [Joh93] many fundamental ideas that were not followed up until much later: elliptical Treemaps, 3-dimensional Treemaps using nested cuboids and cylinders, and hierarchical pie charts called polar Treemaps.

Icicle Plots quote the *Castle Diagrams* [KH81] from the early 1980's as their inspiration. They are common in the statistical sciences and included in many commercial statistics software packages like SPSS. Their main use is to depict cluster dendrograms. Icicle Plots represent the parent-child relationship by vertical adjacency: the child nodes are placed right on top or underneath their parent. A 3-dimensional variant has been envisioned from early on and is also depicted in the initial publication [KL83].

It is these two prominent examples that will mainly be discussed in the following design space definition to illustrate the different approaches.

4.1.1 Dissecting the Design Space

A design space definition identifies independent design dimensions that subdivide and span the design space. These dimensions can then be used to locate existing techniques and to alter them. From the studies of the existing implicit visualization techniques, the following four axes of the design space can be identified – each of which encodes one degree of freedom in the design process:

- **Dimensionality:** 2D or 3D
- **Node Representation:** graphics primitives
- **Edge Representation:** inclusion, overlap, adjacency
- **Layout:** subdivision, packing

[2]Early forms of these diagrams have actually been invented by Gottfried Wilhelm Leibniz (1646-1716) [Bar69]. Yet, their popularization is due to Leonhard Euler (1707-1783) who mentioned them in a correspondence with princess Friederike Charlotte von Brandenburg-Schwedt in 1768. They were used as graphical representations in formal logic and later in set theory. Since any tree can be represented by nested sets, using and deriving these diagrams for tree visualization seems just logical in retrospect.

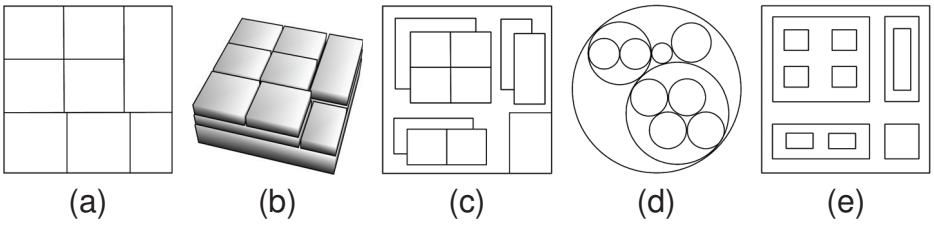


Figure 4.1: Treemap variations to enhance the perception of the tree structure: (a) Original Squarified layout [BHvW00] emphasizing the attribute values, (b) Steptree [BCS04, Bla06] – emphasizing the structure by extrusion, (c) Cascaded layout [LF08] – emphasizing the structure by using overlap, (d) Circular Treemap [WWDW06] – emphasizing the structure by using a non-space-filling primitive, (e) Nested layout [JS91, Joh93] – emphasizing the structure by leaving white space around the primitives.

The latter three properties can have additional parameters that govern the actual realization in detail. These additional parameters are used to fine-tune a visualization. Examples are the surface properties of the primitives (color, texture, transparency,...), edge representation details (amount of necessary overlap to be discernable from inclusion and adjacency,...), and layout constraints (ordering, desired aspect ratio,...).

To understand how these four design dimensions control the visualization outcome, one has to realize the basic conflict that underlies the design of implicit hierarchy visualizations. It is, that these visualizations try to do two things at the same time: convey numerical and categorical attributes of the nodes, as well as their hierarchical relation. This can in principle be done, because both are mapped onto independent visual attributes – the characteristics of the graphical primitives (size, shape, color,...) are used for node attributes, and the relative position of the nodes to each other to encode the parent-child relationship. Yet, often these two design goals of showing attributes and showing structure cannot be fully achieved at the same time. Some implicit tree visualizations are better suited for showing node attributes and thus for supporting attribute-based tasks, while others emphasize on the structure and thus rather assists topology-based tasks.

Examples can be found among the Treemap variations: the *Squarified Treemap* shows a node attribute encoded as node size and facilitates in particular attribute-based comparison tasks by aiming for aspect ratios close to 1 [BHvW00]. Furthermore, by its layout it becomes self-evident that a non-leaf's attribute value is the aggregate of its children's attribute values, because the children occupy the exact same space. While here the hierarchical structure of the data is still present and can be discerned, the focus of this technique lies clearly on representing the attributes of the nodes. On the other end of the spectrum, there are for example *Cascaded Treemaps* [LF08] which put more emphasis on the hierarchical structure. This is achieved by leaving a small unoccupied boundary between a node and its children and by using an overlap relation instead of nesting. Both increase the perception of the actual tree structure, as it conveys a 2.5D impression and maintains partial visibility of lower levels of the visualization (higher levels of the tree). Such variations to enhance the perception of the tree structure are shown in Figure 4.1 for a Squarified Treemap. Here, the original visualization (a) was altered along the four

design dimensions: dimensionality (b), edge representation (c), node representation (d), and layout (e). The individual influences of the four axes and their parametrizations are discussed in the following.

Dimensionality

Dimensionality is highly controversial in information visualization. Most of the published implicit visualization techniques are 2-dimensional. This is only natural, as they borrow from the *map metaphor* (hence Tree**map**) and maps are in general 2D representations. This universal design metaphor allows the visualization user to think of the data in terms of the metaphor, effectively establishing a mental map of the hierarchy. 2D visualizations have several advantages over 3D representations, including that:

- they are **suitable for static media** (e.g., printouts), as no occlusion can occur in 2D that would require interactive view manipulation,
- they are **used in time-critical situations**, as the search space for a node/subtree of interest is only 2-dimensional and no lengthy interactive exploration is needed,
- they **perform better for comparison tasks** on node attributes than 3D techniques, as areas can perceptually be better compared than volumes.

Hence, many people tend to prefer 2-dimensional representations, as they provide higher information availability than 3D [SJOC01]. Most commercial visualization tools and practical implementations on the internet offer 2-dimensional views, if they provide implicit tree visualizations at all.

Yet, it can be seen already from the earliest publications (Icicle Plots [KL83], Treemaps [Joh93]) that 3-dimensional representations were often considered, too. Many later examples like Beamtrees [vHvW02] or Circular Treemaps [WWDW06] do not leave their 3D extensions unmentioned, either. This is again only natural, as 2D representations easily extend to 3D by use of the *cityscape metaphor* [KV97] or the *container metaphor* [Ris08]. They provide an obvious transition path from the 2-dimensional, basic technique to its 3-dimensional extension. Hence, the existing 3-dimensional techniques are usually adaptations or enhancements of a 2-dimensional technique.

So, in the first case in which 3D techniques borrow from the cityscape metaphor, they typically extrude a flat 2-dimensional visualization into the third dimension. This applies to Treemaps as well as to Icicle Plots. Examples for extruded 2D Treemaps are Information Pyramids [AWP97], 3D Nested Treemaps [CKI99], StepTrees [BCS04, Bla06], and even nested Hemispheres [BD04] for circular Treemaps. Icicle Plots have been adapted likewise to 3D Icicle Plots [KL83]. They have in common that they stack node primitives on top of each other, giving the graphical representation a recognizable shape and allowing an intuitive exploration from a bird's-eye perspective.

In the second case in which 3D-techniques rely on the container metaphor, they mostly extend a known 2-dimensional subdivision or packing algorithm to the third dimension. Here only Treemap examples are possible, as Icicle Plots and Sunbursts do not rely on nesting in the first place. The most prominent 3D Treemap examples of this kind are

probably Information Cubes [RG93] and Treecubes [TON03, TON04]. It is apparent that this approach suffers from heavy occlusion and therefore these techniques can only be used with semi-transparent primitives and for rather shallow hierarchies.

For implicit visualizations, the main argument supporting the use of 3D is that implicit techniques rely heavily on spatialization, as they encode the edges of the hierarchy into relative positions. Allowing an additional dimension for this relative positioning opens up new perspectives. A good example for this is the Beamtree technique [vHvW02]. In its 2-dimensional form it is an overlap-technique that encodes the parent-child relationship by overlapping rectangles. As for all overlap-techniques, this leads to occlusion of most of the internal tree structure, leaving mainly the leaves fully visible. Adding a third dimension allows to transform it into a 3D technique that uses adjacent cylinders instead of overlapping rectangles. The orthogonal view from the front still yields a bottom-up view with the leaves up front, resembling the original 2D technique. But in 3D, the whole representation can be rotated and viewed from the back, yielding a top-down view with the root facing the user, or from the side, revealing the individual levels of the hierarchy.

3D extensions of implicit visualizations come into play when it is desired to embed them into 3D applications (volume visualization, virtual collaborative spaces, etc.), or when the visualization designer wants to specifically address the user's spatial memory. Yet, the latter is still somewhat disputed, since there are reports about positive [TL01], neutral [CM04], as well as negative [HONA03] effects of 2D vs. 3D visualizations. But as these comparisons were conducted for visualization techniques in general, it is arguable whether they are applicable to the concrete case of implicit hierarchy visualizations. Nevertheless, notable attempts have been made to make 3-dimensional implicit tree visualizations (usually Treemaps) available to a wider audience. Examples for this come from the field of software visualization – e.g., the award-winning immersive software visualization "CodeCity" [WL07] which uses a Steptree visualization for program code comprehension and the analysis of code evolution and quality, or the Fraunhofer IESE Visual Software Analysis Tool [LHM+09] which likewise uses a 3D Treemap approach for visualizing code quality.

Node Representation

In most practical applications, implicit tree visualizations use rectangles for 2-dimensional representations and cuboids for 3-dimensional views. They are easy to stack, easy to nest, and easy to draw with interactive frame rates even with 10,000s of nodes. There are mainly two reasons for changing the shape of the nodes to something else than rectangles/cuboids.

The first is to improve the aspect ratio of the overall visualization. A common approach is to switch to circles and circle sections: Treemaps become Circular Treemaps [WWDW06], Pebble Maps [Wet03], or Crop Circles [PWG05, WP06], and Icicle Plots become, e.g., PieTrees [DBW00, ODB06], Sunbursts [SCGM00, SZ00], or InterRing visualizations [YWR02] – thus achieving an aspect ratio of 1.

The second reason is to give each individual node a more recognizable shape by using irregular convex polygons. Hence, Treemaps have been altered into Voronoi Treemaps

[AKS+02, BD05] and Circular Partitions [OS08], and the Sunburst technique into the Radial Edgeless Trees [HZH07, HZ07, HZGZ09]. The rather oddly shaped graphical primitives of these techniques are due to their specific layout scheme. These techniques do not center around a certain type of primitive that is then packed as tightly as possible in the available screen space. Instead, the focus lies on the layout algorithm that carves the individual areas of the children out of the screen space occupied by the parent. This carving process yields these irregular polygons. The advantage is a distinct shape for each node that makes the whole visualization less uniform and aids in memorizing and recognizing parts of the structure. The obvious disadvantage lies again in the perception of the node attribute values, as varying shapes are harder to estimate and compare in terms of their area.

Apart from these two shared reasons, Treemaps and Icicle Plots each have a third reason for switching to non-rectangular shapes. For Treemaps, it is the fact that tightly packed rectangles do not leave any of the underlying structure visible. However, packed circles [WWDW06, Wet03] or ellipses [Joh93, ONF07, ONG+07] do, thus enhancing the perception of the tree structure. This way, lower levels stay visible through gaps in between the packing and thus give a better perception of the depth of the tree. But it also leads to the situation that less space is available for each level. As trees tend to grow wider with each level, this aggravates any spatial constraint.

For Icicle Plots, the third reason is a different one: the available space for laying out a node's children stays constant, as the space on top of the root limits the entire layout, independent of how wide the hierarchy actually grows. Yet, this aspect can be lessened by switching to a radial variant with circle sections as primitives like Sunburst. Here the available space (in this case the circumference) grows with increased distance to the root. Different triangular variants follow the same design argument. Notable examples are the Triangular Aggregated Treemap [Chu98], which is a misnomer as it has little to do with the Treemap layout, and Cheops™ [BPV96].

It is also important to note that a 3D extension of a 2D node representation can usually not be determined unambiguously. For example, a circle can be extended into either a (hemi-)sphere, a cylinder, or a cone. And a square can be extended into either a cuboid or a pyramid. For the design space axes to be as independent from each other as possible and not to influence one another, it is important to include the disambiguation in the node representation, e.g., by defining three different circle primitives: circle/sphere, circle/cylinder, and circle/cone.

Additional parametrizations of node primitives may influence the surface properties or rendering style of the nodes. The node surfaces can be used for coloring and texturing to either enhance the perception of structure (Cushion Treemaps [vWvdW99], Cushion Icicle Plots [CAT07]) or the perception of attribute values and changes thereof (Contrast Spiral Treemaps [TS07]).

Edge Representation

In implicit tree visualization techniques, the edges are not represented by a graphical object, but instead encoded by the relative position of the graphical objects that represent the tree's nodes – either by *inclusion*, *overlap*, or *adjacency*. This becomes possible through

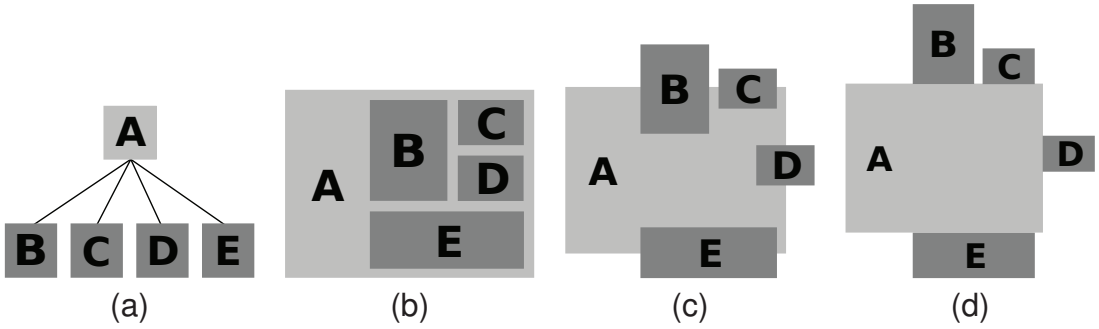


Figure 4.2: (a) Explicit, node-link layout, (b) Implicit layout by inclusion, (c) Implicit Layout by overlap, (d) Implicit layout by adjacency.

the representation of the nodes as graphical primitives that have a spatial dimension and are not just points in space. So, these primitives can for example contain or overlap one another. And by doing so, they implicitly represent the parent-child relation between nodes of a tree as exemplified in Figure 4.2. For this relative positioning of the set of graphical objects O, any spatial relation R can be used that exhibits *acyclicity* (including *irreflexivity*) and *non-convergence*. These conditions make sense, as they directly reflect the structural properties of the tree to be represented:

- **acyclicity:** a node cannot be its own parent, neither directly (irreflexivity) nor indirectly (acyclicity) — $\forall a \in O, \forall n \in \mathbb{N}^+ : \neg(aR^n a)$
- **non-convergence:** a node cannot have two parents, only one – and accordingly two nodes cannot have the same child node — $\forall a \in O \nexists b, c \in O : bRa \wedge cRa$

In addition to these necessary conditions, in some cases it may be useful to also have *asymmetry* or *transitivity*:

- **asymmetry:** if a node a is the parent of a node b, then node b cannot also be a's parent — $\forall a, b \in O : aRb \rightarrow \neg(bRa)$
- **transitivity:** extends the parent-child relationship also to the children's children and so on (ancestor-descendant relationship) — $\forall a, b, c \in O : aRb \wedge bRc \rightarrow aRc$

Both are especially helpful for topology-based tasks. Asymmetry is useful for giving an additional hint on the direction of the edges, pointing from higher levels towards the leaves. This results from the obvious distinction between parent and child, e.g., in the case of node a overlapping node b it is apparent which one is the parent and which one is the child. This is useful when one has zoomed into the visualization and is only able to see a portion of it: even though neither root nor leaves might be visible, it is made obvious by the asymmetry in which direction both lie and how the visualization has to be read. Transitivity on top of that allows to find out whether one node is a (distant) ancestor of another node, as this becomes a simple look-up task. On top of that, it also eases

attribute-based tasks like comparison of node attributes, as with transitivity two related nodes cannot be located too far away, no matter how deep the tree is.

The three commonly used spatial relations fulfill these two conditions, though all differently:

- **inclusion** guarantees asymmetry and transitivity: if a is nested in b, it cannot be the other way around. And if a is nested in b and c is nested in b, it is obvious that c also lies within a.
- **overlap** guarantees only asymmetry: if a overlaps b, b cannot overlap a. On the other hand, transitivity does not hold for overlap.
- **adjacency** guarantees neither asymmetry, nor transitivity, if a is adjacent to b.

The only missing combination is a spatial relation that is transitive and symmetrical at the same time. Yet, it can easily be shown that this case cannot occur, as asymmetry is a necessary condition for transitivity. If it was not $\exists a, b \in O$ with aRb, bRa, which (because of transitivity) leads to aRa, and thus violates the acyclicity/irreflexivity of R.

Adjacency reveals more of the hierarchical structure than inclusion as each individual level is visible and depth equals distance. Yet, inclusion has a very useful property which adjacency does not have: it does not grow outwards and one can be sure that it only uses the amount of space dedicated to display the root node. All other nodes will be placed within this space – and not attached to it. Overlap tries to find a good compromise between both by showing more of the structure, while not growing outwards too much.

As it is mainly the different edge representation that discerns Treemaps from Icicle Plots, there is not much diversity to be expected for this design dimension: Treemap variants use inclusion by default and in very few cases overlap – 2D Beamtrees [vHvW02] and Cascaded Treemaps [LF08]. Icicle Plots and related techniques use adjacency in all cases. Up to now, there seems to be not a single Icicle Plot variant that uses overlap instead – even though this would be a valid option. Overall, it can be stated that inclusion techniques like most Treemap variants are the most common variety among the implicit techniques in use today.

Layout

In general, one can discern two major layout methodologies: *subdivision* and *packing*. Subdivision is applied starting from the root. It takes the space assigned to a node and subdivides it into regions for the children of this node. This method is applied recursively to the next level of the hierarchy. Packing goes the other way around: starting from the leaves, it determines the shapes and sizes of the nodes according to their attribute values and then attempts to pack sibling nodes tightly into their parent's space. Even though packing is known to be NP-complete by reduction to the Bin Packing Problem [GJ79], this is only valid for the optimal solution. If a good approximation is enough, as it is usually the case for visualizations, fast heuristics can be applied for packing the objects. An example is given in [IYIK04] which is used by the visualization method Data Jewelry Box [IKIY02]. Subdivision uses the space fully, whereas packing tends to leave gaps in

between. The pros and cons for each are exactly the same as already discussed for node primitives that cover their ancestors completely (rectangles,...) or not (circles,...): the latter allows a glimpse at the structure through the gaps in between, but sacrifices at each level some of the available space for this.

All known layout methods fall into one of these two categories, even though it may not always be obvious. This is sometimes the case for extruded/stacked techniques. In 2D, these are for example Sunburst or Icicle Plots, which are in fact 1-dimensional subdivision techniques of the perimeter of circle(-segments) or of one side of a rectangle, respectively. Also in 3D, this can be observed for Steptrees [BCS04, Bla06] and 3D Circular Treemaps [WWDW06] which are actually a 2-dimensional subdivision and packing technique, respectively. This shows that the user always has more than one option if the tree structure needs to be emphasized: instead of changing the layout from subdivision to packing, one can still maintain the subdivision and extrude the layout into another dimension to achieve the same goal.

Additionally, it is usually the layout algorithm that has to ensure that certain external constraints are met. As already mentioned in Chapter 2.4.2, Wattenberg [Wat05] introduced four fundamental constraints to be met by a "perfect" implicit layout:

- **stability**: small changes in the data should only result in small changes in the visualization,
- **split neutrality**: structural changes should only affect the parent-region they occur in,
- **order adjacency**: if defined, an ordering of the nodes should be maintained,
- c**-locality**: the representation should be as compact as possible (aspect ratio ≈ 1).

While, e.g., Icicle Plot variants handle the ordering of nodes by design, other constraints must be explicitly enforced in the layout. Different layouts put a different emphasis on these constraints. Some of them are mostly concerned with order adjacency (Spiral Treemap layout [TS07], Ordered Treemap layout [SW01, BSW02]), others with c-Locality (Squarified Treemap [BHvW00]). Wattenberg introduced the *Jigsaw Map* [Wat05] as one possible realization that fulfills all of the above constraints.

Apart from that, the layout may be required to fulfill additional application-specific demands. This is the case, e.g., for a *2-dimensional order adjacency* as realized for cartographic contexts in Spatially Ordered Treemaps [WD08], or when dealing with *unusual aspect ratios* as in TreemapBars [HHZ09], which embed Treemaps into bar charts.

Even though the layout of the implicit visualization is introduced as an independent dimension of the design space, it is notable that there can be some interplay with the choice of the node representation. While packing layouts can (in theory) handle all kinds of primitives, this can get complicated for subdivision layouts. Because in this case, it is the layouts that actually shape the node representation by subdividing the space. So, an independent choice of node primitive and layout is hardly possible. If a shape is chosen other than the one that is generated by the layout, it must be scaled and packed inside the space generated by the layout.

4.1.2 The Design Space as a Whole

Bringing the discussed design dimensions together to form the actual design space raises several concrete questions that are worthwhile to investigate. First and foremost, the question is whether the overall design space is complete and consistent, two important properties a design space should fulfill. In addition, the required practical usability of the design space beyond the pure classification of the existing techniques imposes additional design issues which are also discussed in the following.

Completeness and Consistency

Telling when a design space definition is *complete* is a complicated task. There may very well be undiscovered or unpublished implicit hierarchy visualizations that are consistent with the design space definition, but that are so awkward that the proposed design space definition does not include them (yet). So since it is hard to tell definitely whether a design space is complete or not, a design space definition is assumed to be complete, if it contains the known implicit techniques as of today. This means it may need to be expanded in the future if an implicit visualization is found that is not yet covered by the design space. While completeness is hard to prove, Table 4.1 shows that it is possible to position all visualization techniques discussed so far in the design space. These visualization techniques are also shown in Figures 4.3 and 4.4, which comprises the majority of today's implicit hierarchy representations. Even though these are of course not all existing implicit techniques, being able to place these visualizations in the design space is already a solid indication for the design space's completeness.

While an incomplete design space would be the result of a design space definition that is too rigid, an inconsistent design space usually results from a definition that is too broad and may go well beyond the design space originally intended. A design space definition for implicit techniques is considered as *consistent*, if it does not violate the basic design principle of implicit visualizations: representing the hierarchy without drawing edges. This excludes two kinds of visualizations:

- **any dissociate arrangement** of graphics primitives that does not represent the hierarchical structure at all, as there is no apparent spatial relation between them. An example is the Graph Signature Visualization [WFC+06] shown in Figure 4.5(a). This visualization uses a scatterplot to show different node classes. Even though they do not include any links between the nodes, these visualizations cannot be considered implicit, because the hierarchical structure is completely lost in the mapping step and cannot be discerned from the representation any more.

- **any visualization that contains explicit links**, e.g., Information Slices [AH98] depicted in Figure 4.5(b). Even though they are an implicit visualization techniques at their core (a semi-circular Sunburst variation), not unlike Elastic Hierarchies [ZMC05] they use explicit links to connect multiple, otherwise separate parts (slices) of the implicit hierarchy visualization and are thus not a part of the design space.

	2D	3D	Adjacency	Overlap	Inclusion	Rectangles/Cuboids	Squares/Pyramid (Frustums)	Triangles/Trapezoids	Polygons	Circle(-segments)/Cylinders	Circle(-segments)/Spheres	Ellipses/Ellipsoids	1D Subdivision	2D Subdivision	3D Subdivision	1D Packing	2D Packing	3D Packing	Year Published	Figure
Treemap [JS91],[Shn92]	■				■	■								■					1991	4.3(d)
Contrast Spiral Treemap [TS07]	■				■	■							■						2007	4.4(q)
2 1/2D Treemap [TJ92]	■	■			■	■	■							■					1992	4.3(e)
Polar Treemap [Joh93]	■				■					■	■			■					1993	4.3(f)
Jigsaw Map [Wat05]	■				■				■				■						2005	4.4(g)
Generalized Treemap (Pie) [VvWvdL06]	■				■					■	■			■					2006	4.4(j)
Generalized Treemap (Pyramid) [VvWvdL06]	■				■			■						■					2006	4.4(k)
Generalized Treemap (Pie+Pyr.) [VvWvdL06]	■				■					■	■			■					2006	4.4(l)
Tree Cube [TON03],[TON04]		■			■	■									■				2003	4.4(c)
3D Treemap [Joh93]		■			■	■									■				1993	4.3(i)
Cushion Treemap [vWvdW99]	■				■	■								■					1999	4.3(n)
Voronoi Treemap [BD05]	■				■				■					■					2005	4.4(f)
Circular Partitions [OS08]	■				■				■					■					2008	4.4(s)
Quantum Treemap [Bed01]	■				■	■								■			■		2001	4.3(r)
Data Jewelry Box [IKlY02]	■				■	■											■		2002	4.3(s)
Cascaded Treemap [LF08]	■			■		■								■					2008	4.4(r)
Ellimap [ONG+07],[ONF07]	■				■							■		■					2007	4.4(m)
Treemaps with Ovals [Joh93]	■				■							■		■					1993	4.3(g)
Pebble Map [Wet03]	■				■					■	■						■		2003	4.4(b)
CropCircles [PWG05]	■				■					■	■						■		2006	4.4(h)
Lifted Treemap [CS09]		■			■	■								■					2009	4.4(t)
3D Nested Treemap [CKI99]		■			■	■								■					1999	4.3(o)
Information Cube [RG93]		■			■	■												■	1993	4.3(j)
Nested Columns [Joh93]		■			■					■							■		1993	4.3(h)
2D Icicle Plot [KL83]	■		■			■							■						1983	4.3(b)
Castles [KH81]	■		■					■					■						1981	4.3(a)
Cushioned Icicle Plot [CAT07]	■		■			■							■						2007	4.4(n)
Triangular Aggregated Treemap [Chu98]	■		■					■					■						1998	4.3(m)
Sunburst [SZ00]	■		■							■	■		■						2000	4.3(q)
Interring [YWR02]	■		■							■	■		■						2002	4.3(t)
PieTree [DBW00]	■		■							■	■		■						2000	4.3(p)
Radial Edgeless Tree [HZH07],[HZGZ09]	■		■					■					■						2007	4.4(o)
Steptree [BCS04]		■	■			■								■					2004	4.4(d)
3D Icicle Plot [KL83]		■	■			■							■						1983	4.3(c)
Nested Hemispheres [BD04]		■			■						■						■		2004	4.4(e)
3D Nested Cylinders and Spheres [WWDW06]		■	■							■	■						■		2006	4.4(i)
3D Sunburst [SKW+07]		■	■							■				■					2007	4.4(p)
3D Beamtree [vhvW02]		■	■		■					■			■						2002	4.4(a)
Information Pyramids™ [AWP97]		■	■				■							■					1997	4.3(j)
Cheops™ [BPV96]	■		■					■								■			1996	4.3(k)

Table 4.1: Classification of most existing implicit tree visualization techniques, approximately ordered from the most attribute-centric technique at the top to the most structure-centric one at the bottom.

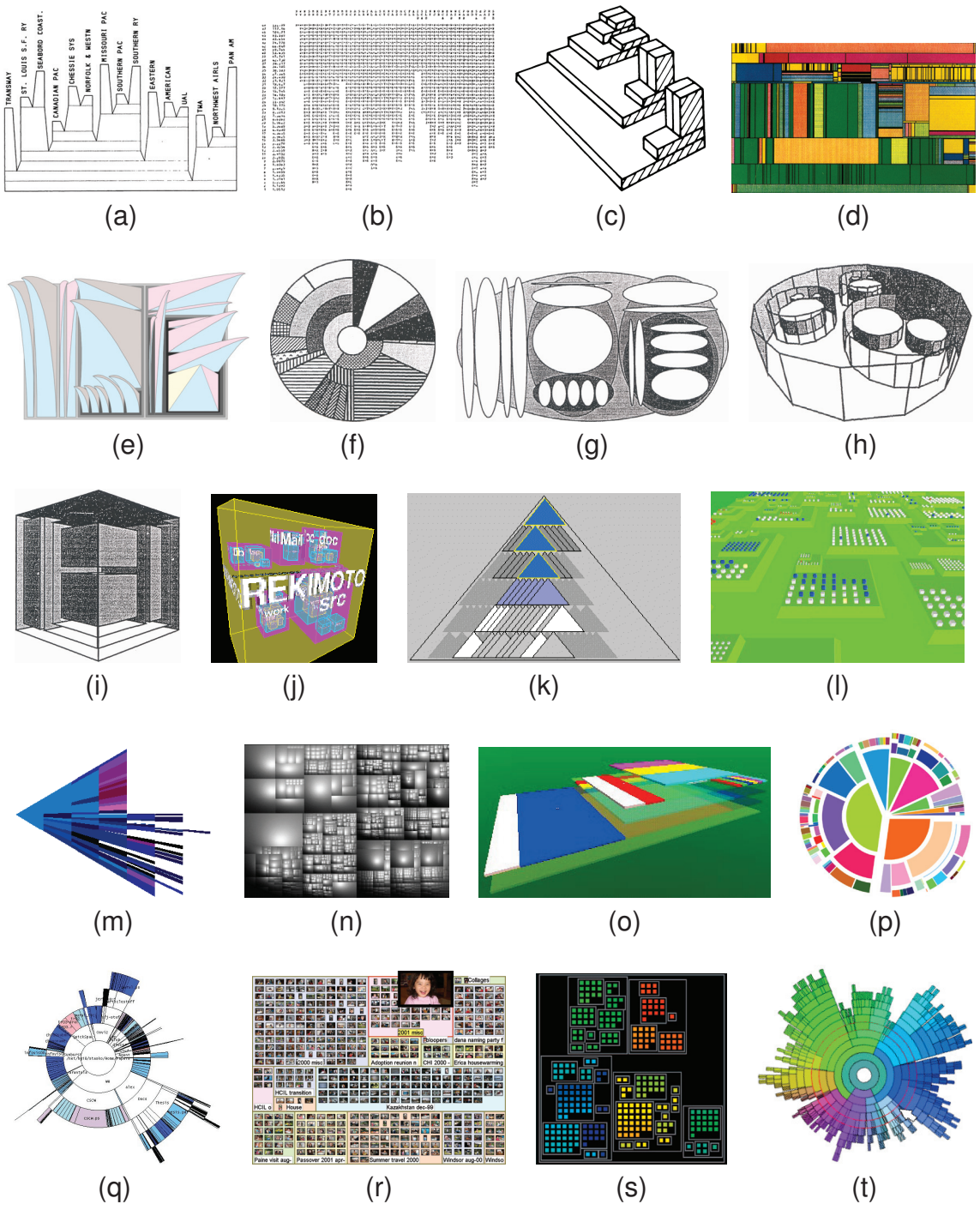


Figure 4.3: A selection of implicit tree visualizations. Part A (1981–2002): (a) Castles [KH81], (b-c) 2D+3D Icicle Plots [KL83], (d) Treemap [JS91, Shn92], (e) 2 1/2-D Treemap [TJ92], (f) Polar Treemap [Joh93], (g) Treemap with Ovals [Joh93], (h) Nested Columns [Joh93], (i) 3D Treemap [Joh93], (j) Information Cube [RG93], (k) Cheops™ [BPV96], (l) Information Pyramids™ [AWP97], (m) Triangular Aggregated Treemap [Chu98], (n) Cushion Treemap [vWvdW99], (o) 3D Nested Treemap [CKI99], (p) PieTree [DBW00], (q) Sunburst [SZ00], (r) Quantum Treemap [Bed01], (s) Data Jewelry Box [IKIY02], (t) InterRing [YWR02]

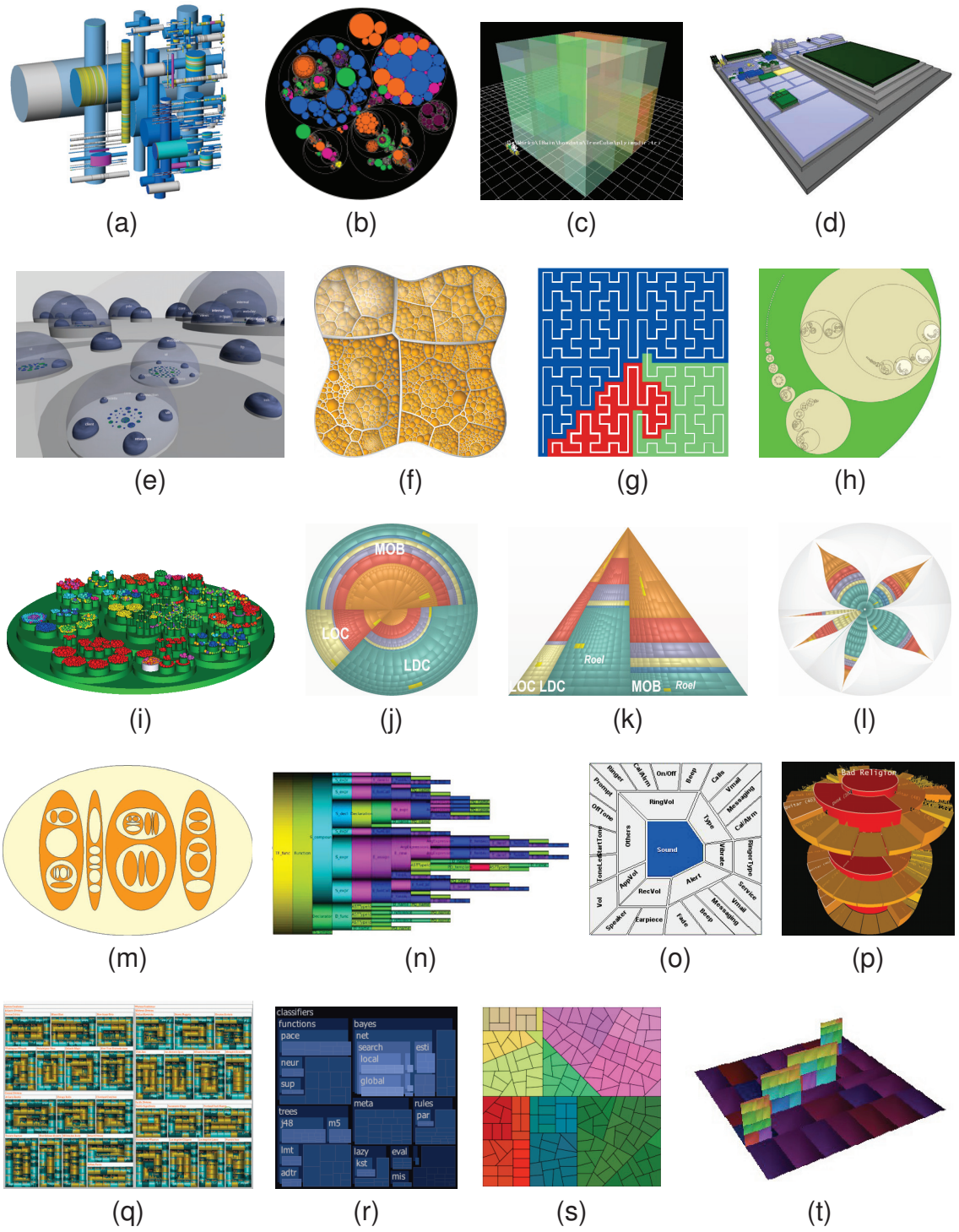


Figure 4.4: A selection of implicit tree visualizations. Part B (2002–2009): (a) 3D Beamtree [vHvW02], (b) Pebble Map [Wet03], (c) Tree Cube [TON03], (d) StepTree [BCS04], (e) Nested Hemispheres [BD04], (f) Voronoi Treemap [BD05], (g) Jigsaw Map [Wat05], (h) CropCircles [PWG05], (i) 3D Nested Cylinders and Spheres [WWDW06], (j-l) Generalized Treemaps [VvWvdL06] – pie, pyramid, and combination of both, (m) Ellimap [ONG+07, ONF07], (n) Cushioned Icicle Plot [CAT07], (o) Radial Edgeless Tree [HZH07, HZGZ09], (p) 3D Sunburst [SKW+07], (q) Contrast Spiral Treemap [TS07], (r) Cascaded Treemap [LF08], (s) Circular Partitions [OS08], (t) Lifted Treemap [CS09]

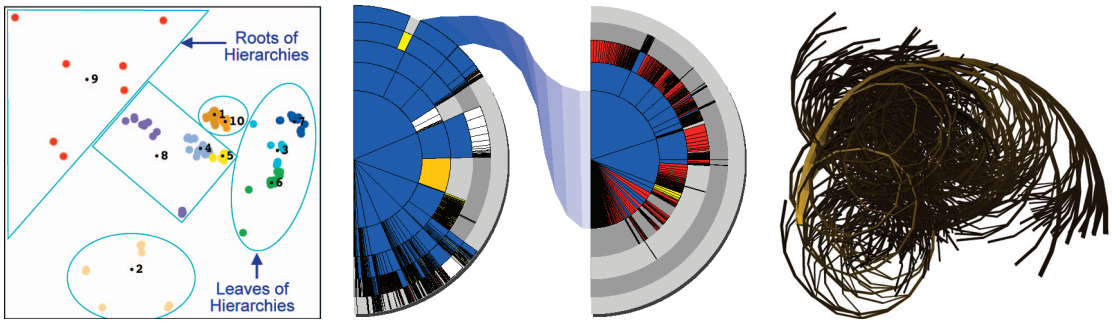


Figure 4.5: Examples from the boundary of the design space: (a) Graph Signature Visualizations [WFC+06] and (b) Information Slices [AH98] are examples of visualizations not covered by the design space, whereas the Strip Treemap variant of a Spiral Steptree (c) may not be a very useful visualization, but is nevertheless part of the design space (cp. Figure 4.11).

The consistency is basically ensured by the way the design space is constructed as it does not provide any explicit edge representations (fulfilling the second condition) but forces the layout to use at least one of the available edge representations (adjacency, overlap, or inclusion) and to fulfill acyclicity and non-convergence. This effectively prevents any "dissociate arrangement" and arrangements that do not represent a tree (fulfilling the first condition).

Having a complete and consistent design space does not prevent the definition of mostly "useless" visualization techniques. Figure 4.5(c) shows an example of such a visualization design that is so heavily occluded by its own layout that it obscures more of the data than it actually shows. This is a problem that all vast design spaces face: it is easy to get lost in them and hard to find the subspaces of useful techniques. Yet, if the designer wants to explore only the known design decisions, a list of the existing techniques as it is given in Table 4.1 would be sufficient. So, it is not seen as a burden when giving the designer the ability and flexibility to roam freely in the design space, but rather as a chance to discover completely new techniques. Actually, it can be very educational to see when and why well-known techniques break down when altering them along certain design dimensions. It may be even the case for special types of input data (e.g., only for binary trees of bounded height), that formerly useless techniques "degenerate" into expressive, efficient, and effective visualizations. And this can be taken further to actually tailor a visualization specifically to a concrete data set and task at hand by incremental refinement of its design.

Another implication of a complete design space is that it allows to define impossible combinations of design parameters, for which no reasonable practical realization exists. This occurs when at least two of the made design decisions are incompatible – e.g., when the area provided by the layout method and the (base) area of the child do not match. This is always the case when one does not respect the dependencies between layout and node representation mentioned at the end of Chapter 4.1.1. An example would be the use of

a Voronoi Treemap layout with circles instead of the convex polygons provided by the layout. This example is not far fetched, as similar, incompatible combinations of graphics primitives are frequently used under the term "Circle Maps" in cartography. Since the flexible design space definition does allow such incompatible combinations, a way must be found to handle these. Because of the sheer number of possible but incompatible combinations, this problem is far too complex to be solved elegantly in the conceptual design space. Instead, a practical solution would be to require each node primitive to provide mechanisms to adapt to a possibly incompatible area.

In this concrete case, a suitable solution would be that each node primitive must provide methods that compute an inscribed as well as a circumscribed circle/sphere and rectangle/cuboid. They are used to interface between a node primitive and an incompatible primitive provided by a layout. This is done by inscribing into the primitive generated by the layout and by circumscribing the node primitive – both with the same interface: either circle/sphere or rectangle/cuboid. They can then easily be put together. The inscription mode to use (circle/sphere or rectangle/cuboid) is the one that maximizes the resulting area/volume occupied by the inscribed primitive. This pragmatic approach does not guarantee an optimal usage of the space and may not in all cases lead to aesthetically pleasing results. Yet, it effectively bridges the gap between the general design space and a prototypical realization that allows to put together new visualization techniques.

Design Issues

To be applicable as a theoretical framework for the classification of visualization techniques and as a practical tool that aids in the design of new visualization, a number of issues are worthwhile to be mentioned:

Mixing Design Choices. The definition of the design space has so far in all examples been applied to the tree as a whole. Yet, the restriction to a global scope for design decisions is neither necessary nor useful. In fact, there are a number of *mixed implicit visualizations* that cannot be expressed without a more refined, local way to specify the visualization design. The term *mixed* was first used for *mixed Treemaps* by Vliegen et al. for their Generalized Treemaps [VvWvdL06] in which the authors use a Slice-and-Dice layout for the upper levels of a hierarchy and the squarified layout for the lower levels. The concept of mixing different design choices, be it layouts or node primitives, has been known (but not been named) well before that. It can also be observed for:

- **2 1/2D Treemaps** [TJ92] mixing a 2D representation for the inner nodes with 3D pyramid-like graphics objects for the leaves
- **Quantum Treemaps** [Bed01, BSW02] mixing a subdivision layout for the inner nodes with a packing layout for the leaves
- **3D circular packing** [WWDW06] mixing cylinders for the inner nodes with hemispheres for the leaves
- **2D Beamtrees** [vHvW02] mixing overlap for the inner nodes with inclusion for the leaves

- **3D Beamtrees** [vHvW02] mixing adjacency for the inner nodes with inclusion for the leaves

It is apparent from this list that differing design choices for leaf- and non-leaf-nodes are the most common form of mixing. This is either because they are of different types (e.g., directories vs. files) or just to help discerning them. As it is mentioned in [VvWvdL06] and [SDW09], design choices can be made on a per-node basis. This does not only allow to map structural node properties (is_leaf, depth,...) to design parameters, but also derived properties like Strahler numbers [ADD+04] or other numerical or categorical attributes. Mixed representations can be used to emphasize entire subtrees, as it is done in [CS09] by adding colorful, orthogonal sub-Treemaps on top of a regular 2-dimensional "base Treemap".

Positioning in the Design Space. While the design dimensions are sufficient to be used for general discussions, for a practical application they need to be further specified. The reason is that they allow to classify an individual technique conceptually, but they are not concrete enough to define each and every aspect of that technique, which would clearly position the technique in the design space. And this is only natural, as no design space is defined with only its practical applicability in mind, but it should first and foremost serve as a clear mental map for a class of visualizations. Detailed parameters that manage every tiny facet of a visualization's appearance would only obfuscate this map. Yet, for a concrete realization of the design space these additional clarifications are needed to resolve otherwise unspecific design decisions. An example is that the choice of a node primitive does not specify where exactly to attach the child nodes in case of adjacency – e.g., for a cylinder it can be either its flat top (3D Circular Treemap [WWDW06]) or its curved coat (3D Beamtree [vHvW02]). So, before the design space can actually be realized, it must be adapted by adding missing details like this to its definition. In case of attaching 3D primitives, one needs to specify three additional attributes for the primitives: the surface onto which to attach the children, the surface normal to indicate which side to attach to, and the anchor point where to connect with the parent.

Ranges in the Design Space. The design space can be used to trace different design alternatives across the entire design space. One example for such a design alternative would be the mentioned main design conflict between attributes and structure. As it is often not possible to decide for either one of these options, the resulting visualization should rather be a good compromise between both. Techniques that focus solely on either one can be understood as the endpoints of this particular range towards which an implicit visualization can be pushed in the design space to achieve its goal. A prominent application example for a visualization that focuses almost entirely on the attributes instead of the hierarchical structure is Wattenberg's *Map of the Stock Market* [Wat99]. In this case, the focus is only natural, as the hierarchy in question has only three levels and is easy to overview: the whole stock market (root), the industrial sectors (internal nodes), and the individual companies (leaves). Representatives of both extremes are:

- **focus on attributes: Squarified Treemaps** – a 2D inclusion technique using rectangles of aspect ratio close to 1 and a space-filling subdivision layout

- **focus on structure: Cheops™** – a 2D adjacency technique that uses overlapping triangles of uniform size and a nested packing technique with lots of white space to visualize paths and subtrees

Within the range between these endpoints lie all the intermediary compromises between showing values and showing structure. Table 4.1 is approximately sorted along this spectrum with the attribute-oriented (squarified) Treemap technique at the top and the structure-oriented Cheops™ at the bottom. Another example would be the range of how siblings are spatially related to each other: all the way from no apparent spatial relation (as in the original Treemap [JS91, Shn92, Joh93]), through a certain degree of co-location (as in Wattenberg's Stock Market Treemap layout [Wat99], later called Cluster Treemap [BSW02]), to a linear order (as in Jigsaw Maps [Wat05] or Spiral Treemaps [TS07]) and finally a direct overlap (as in Cheops™ [BPV96]).

Displacement in the Design Space. The concept of a design space with the individual visualization techniques being concrete positions within it allows to establish a notion of similarity between different techniques. As the term "design space" just serves as an analogy, it is hardly possible to define a distance metric for the design space in the mathematical sense. Yet, each change in design decisions can be understood as a displacement in the design space, which in turn provides a measure of similarity. So, the number of changes needed to convert one technique into another gives a sense of how much they have in common and thus how close they are conceptually – very much like edit distances. This provides a possibility to directly develop visualizations that are either especially close or rather dissimilar to a given technique. This is quite useful for devising user studies for a newly developed technique, as it allows to find established techniques that are close to the new technique as candidates for comparison. As an example, the conversion path between 2D Polar Treemaps and Information Pyramids shows that changes on all four design axes are needed to transform one into the other. While this might not be the shortest or most direct path within the design space, it is one that only uses known techniques as intermediary steps:

1. **2D Polar Treemaps:** change relationship from nesting to adjacency – yields: 2D Sunburst
2. **2D Sunburst:** change primitive from circle(-section) to rectangle – yields: 2D Icicle Plot
3. **2D Icicle Plot:** change dimensionality from 2D to 3D – yields: 3D Icicle Plot
4. **3D Icicle Plot:** change layout from horizontal slicing to squarified – yields: Steptree
5. **Steptree:** change primitive from cuboid to pyramid frustum – yields: Information Pyramids

It is this notion of a displacement corresponding to a stepwise layout change, that serves as the basis for the design space implementation presented in the following second

part of this chapter. It transforms the abstract design space into a rapid visualization prototyping software in which each design change of the visualization translates exactly into a displacement along one of the introduced design dimensions, exactly as in the example above.

4.2 Realizing the Design Space

Since it was the aim from the beginning to raise awareness of the implicit tree visualization design challenges by providing a hands-on experience, it is only logical to provide not only an abstract design space, but an actual tool to do so. A software implementation of the design space can help to define new visualizations in the spirit of *Rapid Visualization Prototyping* by simply exploring new design parameter combinations. Yet, to achieve this, a concrete realization has to go beyond the mere design space and cover the entire visualization pipeline from data input and preprocessing, through mapping and rendering, all the way to its interactive exploration. These steps are shortly described in the following and the resulting prototype is introduced. The concrete software realization of this concept was programmed by Steffen Hadlak during his time as a student assistant in the graduate school "dIEM oSiRiS".

4.2.1 Data Input and Preprocessing

To try out new techniques on real data or to even generate tailor-made visualizations to certain data sets or a certain kinds of data (e.g., binary trees), it is important to be able to load custom hierarchies. The design space implementation supports the import of CSV- and TreeML-formatted [FP03] hierarchies, as they are the most widely used. The CSV-format is the same as used by ManyEyes [VWvH+07], so that all hierarchy data sets from the ManyEyes web site can be used as well. On top of that, as it appears to be a frequent use case for implicit tree visualization techniques, it is also possible to import the directory tree from a local drive or folder. To be able to reproduce such a local directory tree on a different machine, the user can export it to TreeML to pass it on. Two benchmark hierarchies have been included directly with the software in case no other data set is available: the Java 1.6 class hierarchy (2,971 nodes) and the mammals subtree of the InfoVis 2003 contest classification hierarchy "A" (3,023 nodes). Both data sets are relatively small in order to keep the web-based applet to a downloadable size.

Preprocessing (e.g., normalizing numerical node attributes or computing Strahler numbers) and filtering operations are realized as adaptable and exchangeable scripts. They take a tree's root node as input parameter and then traverse the tree in any given order and decorate the nodes with attributes. Some simple structural node attributes are already precomputed, others requiring more computational overhead are defined via Groovy scripts that can be run and thus attached to nodes on demand. The attributes included with the software are:

- **precomputed attributes**: depth, number of children, siblings, depth of subtree, size of subtree (node count)

- **scripted attributes**: Strahler numbers [ADD+04], eccentricity values [RP89]

Custom attributes can likewise be defined through Groovy scripts, which have access to all features provided by the Groovy scripting language. So, it is possible to retrieve additional node attributes from a database or a web service if needed. These scripts are instances of "descriptive measures", a concept that is discussed in more detail in Chapter 6.1.

4.2.2 Mapping

At the heart of the implementation stands the mapping as the visualization step that incorporates the individual design decisions into the overall representation. It is this step that actually realizes the described design space. To do so, a state/operator concept is utilized – independently developed from, but not unlike the one used in the *HiVE system* [SDW09]. It defines the specification of an implicit visualization design as a sequence of operators altering an initial visualization (usually an established technique like Treemap or Sunburst) into a custom visualization. Each operator is hereby described through a 5-tuple *(axis, decision, parameter, function, scope)* with the individual elements being:

- The **axis**-entry names one of the four basic design dimensions introduced in Chapter 4.1.1, identifying which design dimension is affected.
- The design **decision** names a concrete implementation for the chosen axis – e.g., for the design axis "layout" this can be any out of Slice-and-Dice, Circle Packing, Squarified, Strip, etc.
- Additional **parameters** specify details for the chosen design decision, e.g., the border width for a nested layout or the height of a stacked cuboid node representation.
- **Functions** assign the aforementioned parameters a value, either in the form of a constant or as more sophisticated function deriving the value from other properties.
- The **scope** finally defines the set of nodes, for which this specification should be applied, e.g., only leaves, only nodes of a certain depth, or only a certain subtree.

A plain language example taken from the specification of the Treecube technique would be *(node representation, rectangle/cuboid, transparency,* 0.8×*depth, all nodes)* which assigns all nodes (scope) a rectangle/cuboid (decision) as node representation (axis) with a transparency (parameter) of 0.8×node depth (function). This is still a simple design operator, but it illustrates how the abstract design space definition transforms easily into concrete specification steps. Hence, the mapping step is simply executed by applying a sequence of such mapping operators to an implicit visualization chosen as the initial state.

4.2.3 Rendering

As individual visualizations may have special rendering needs, the rendering process can be parameterized in many ways through the user interface. These parameter adjustments

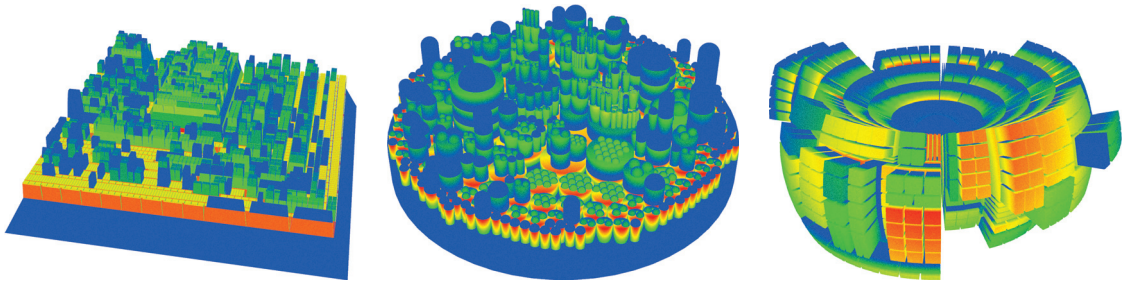


Figure 4.6: Non-photorealistic renderings of the Java 1.6 class hierarchy using (a) Information Pyramids, (b) 3D Circular Treemap, and (c) a 3D Sunburst from Chapter 4.3.1

include: lighting direction, light color, overall brightness, perspective/orthogonal projection, etc. These parameters go beyond the actual design space, but are equally important for the final appearance of a visualization, as several design choices implicitly depend on matching rendering options. For example, a Treemap with textures and bump mapping [HVvW05] needs a suitable lighting to actually show the differences between the node surfaces.

Basically, the individual ways of rendering a design as defined through interactive parametrization can be seen as different "interpretations" of the design space. So, while the mapping and thus the design stays the same, the use of different shaders and rendering styles results in different presentations of the made design choices. An example is shown in Figure 4.6, where a non-photorealistic "heatmap shader" was used. It colors the primitives using a "temperature parameter", which in this case is set to the number of siblings: the fewer siblings a node has, the colder its primitive (e.g., the root having no siblings is very cold) and the more siblings it has, the warmer it gets. This shader omits any ambient light and renders primitives as if they emit light on their own, depending on their temperature. The idea behind this is that with no shadows or glares, the surface structure of a 3D visualization can be perceived equally well from any angle. Another useful rendering style is for example the so-called *Ghost View* [LS08] which ensures the visibility of otherwise occluded nodes of interest.

4.2.4 Interaction

A prototyping toolkit for graph visualization differs from graph visualization toolkits in that regard, that it is not centered around the inspection of a given data set, but around specifying and modifying a visualization technique. Hence, all interaction is geared towards that end and interaction techniques specifically for exploring graph data, such as dynamic filtering [TAS09], are not included in the toolkit. Yet, filtering can nevertheless be realized as part of the interactive visualization design by mapping all nodes that do not fit a certain criteria to transparent primitives.

The interactive specification of parameter settings for a visualization stands at the center of the interaction concept. This includes all three steps needed to define a visualization:

the execution of suitable preprocessing steps, the assignment of mapping-tuples to (parts of) the tree, and the parametrization of the rendering.

The **execution of preprocessing steps** is needed to compute node attributes derived from existing attributes or from structural properties. These derived attributes have the ability to capture properties that go beyond the individual node and express aspects of entire subtrees (average number of siblings, etc.), which can in turn be used for sophisticated mappings afterwards. As not all possible derived attributes are needed all the time, their computation is triggered interactively. If one computation depends on another computation to be executed before, lists of preprocessing steps can be defined that will be executed exactly in the given order.

The **assignment of mapping-tuples** is probably the step that offers the most engaging interaction, as it works in a straightforward point&click fashion. To achieve this, the GUI as well as the visualization preview offer a rich set of controls to specify the mapping-tuples without even being aware of them. If a specific parametrization for a certain node or subtree is needed, the scope can be interactively redefined by selecting a node and then altering its appearance by manipulating the GUI controls for the individual design aspects – e.g., selecting a different node primitive or layout. To locate the node to be selected and subsequently altered, the visualization can be navigated via zoom, pan, and rotation (for 3D). All interactive changes to the specification are logged as a list of the said 5-tuples, which can be stored, passed on, and reused. To explore design alternatives, the specification process can be split at any time by "cloning" the current view, which also establishes individual lists of mapping-tuples for each view. Such a split can be seen in Figure 4.7, where two separate views showing the same data set have been further specified independent from each other and thus become completely different visualizations – sharing only the same dimensionality and color scheme.

The **parametrization of the rendering** includes mostly the adjustment of lighting direction and color, as well as the decision for one of the different rendering styles, such as wireframe or non-photorealistic using the "heatmap shader". Both need to be considered in conjunction and setting them interactively allows to co-design both until a good combination is achieved. To inspect the influence of a certain light setting to different regions of the layout simultaneously, it is possible to sync multiple views, for example to always show the otherwise invisible back side of a 3D visualization in a second view. The effect of shadows and occlusion can be investigated by interactively hiding subtrees and bringing them up again through collapse and expand operations.

4.2.5 Prototype

A prototype of the software implementation is accessible online as a Java applet[3]. It utilizes JOGL for 3D graphics and Groovy for the scripting support. The software is equipped with a number of the most common primitives, layouts, and preprocessing algorithms in this field, so that it allows to build the largest part of the existing implicit hierarchy visualization techniques out of the box. Many new techniques can already be generated just by re-combining and mixing those well known design decisions in new and

[3]http://vcg.informatik.uni-rostock.de/ hs162/itvtk/start.html

unexpected ways – some of them exemplified in the following third part of this chapter. The software shown in Figure 4.7 displays different implicit visualization prototypes in multiple linked views. The linked views have proven to be extremely helpful for comparative testing of different techniques. This makes the software discussed here a versatile and powerful tool for rapid visualization prototyping of implicit hierarchy visualization techniques. The following third part illustrates its flexibility by using it to derive novel visualization techniques along the lines of three different design strategies.

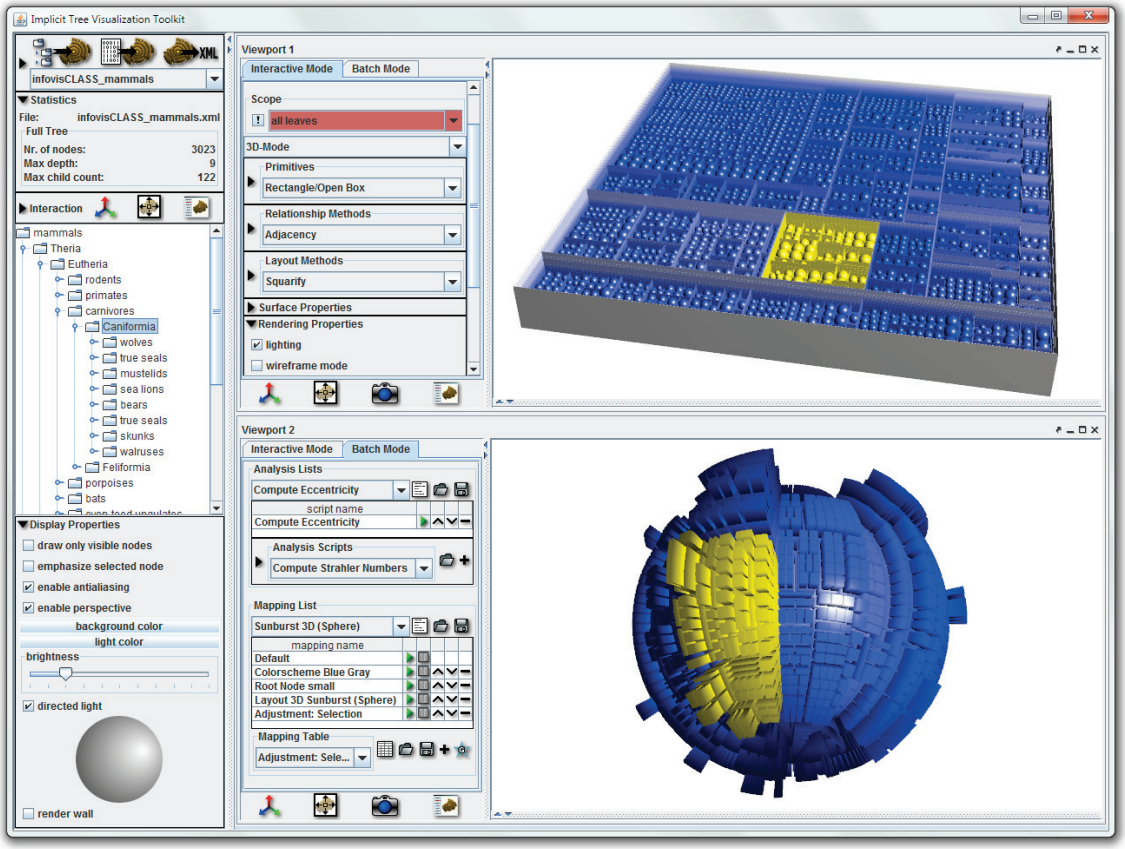


Figure 4.7: The Implicit Tree Visualization Toolkit – The figure shows the mammals of the InfoVis 2003 contest classification hierarchy "A" with the subtree "caniforms" (dog-like animals) highlighted in all views. On the left-hand side, our software shows statistics of the data set, a regular tree view, and controls for visualization parameters that have a global effect. Multiple views can be opened to create implicit tree visualization prototypes. Each view has a set of local controls to alter visualization parameters on a per-view basis either by interactive adjustment or via scripting. Parameter settings and scripts can be stored for later reuse. The top view shows a mixed 3D Treemap technique that stacks boxes inside one another for internal nodes and represents leaves as spheres inside these boxes. This unobtrusively accentuates the leaves and makes this prototype look very much like a carefully arranged collection of items, in this case all sorts of mammals. The bottom view shows a spherical Sunburst variant as discussed in the following part.

4.3 Using the Design Space

With the awareness of the overwhelming possibilities of implicit tree visualizations and the availability of the prototyping tool that allows to access and utilize this vast space of visualization solutions, a user is now enabled to go beyond applying existing standard visualizations to scenarios that actually call for more tailored solutions. With the implemented design space at hand, a user can create and fine-tune a visualization to achieve an expressive, efficient, and effective representation for a given task on a given data set – regardless whether it already exists or not.

This part highlights three important design strategies that realize the prototyping of such tailored implicit tree visualizations. In many cases this leads to useful and aesthetically pleasing, novel visualization results. All three of these strategies are independent of one another, so that they can also be used in conjunction if needed:

- **combine** suitable positions along the design axes into an appropriate visualization – be it new or (by chance) already existing,
- **mix** suitable visualizations by applying them to different parts of the tree – e.g., an attribute-centric visualization for the leaves and a topology-centric visualization for other nodes,
- **parametrize** a found visualization design by enriching it with derived measures – e.g., some of the attributes computed by the Groovy scripts or by changing the rendering style.

Each of them is illustrated by a few examples of novel visualization techniques in the following, filling some of the gaps in the design space that have been left unexplored and unpublished so far.

In these examples, the focus lies on different node primitives, as a large part of a visualization's appearance is defined by the used graphics primitives. In many cases, in which node primitives apart from the standard 2-dimensional rectangle are used, the techniques are named after that primitive: Information **Pyramids**, Information **Cube**, Tree **Cube**, Crop**Circles**, **Elli**maps, etc. And as pointed out in Chapter 4.1.1, a number of design considerations can be addressed just by the choice of an appropriate primitive. So, this chapter discusses mostly design aspects of the node representation along the lines of the three points listed above: combining different node primitives with choices of other design dimensions, mixing different node primitives, and parametrizing node primitives according to derived measures.

4.3.1 Combining Node Primitives with Choices of Other Design Dimensions

This design strategy sees an implicit visualization technique as a combination of values of the individual design axes. Hence, it assumes that a tailored visualization can be achieved by suitable, individual design choices along each of the design axes. The more of such

choices are available, the more likely it is to find one that is suitable for a given problem. Keeping the focus on the node primitives, this part picks a few noteworthy concrete techniques from the so far largely unexplored design subspace of spherical and cylindrical node representations. These exemplify, how easy it actually is to combine these primitives with different choices of dimensionality, edge representation, and layout into novel visualization prototypes.

As a prerequisite for actually allowing the combination of these primitives with all other visualization parameters (edge representation, layout, etc.), it is necessary to make all possible cut sections of these primitives available. Otherwise, e.g., a Slice-and-Dice layout combined with these primitives will not work as expected. The 3-dimensional sections for the following two examples are shown schematically in Figure 4.8, which lists all the different polar cuttings of a cylinder and a sphere. Sphere and cylinder sections are not new, but apart from the 3D cylindrical Sunburst [SKW+07] they where so far unexplored in the context of implicit tree visualizations. Once made explicit like this, these sections are no longer just parts of a standard primitive (cylinder, sphere), but they are parametrizable primitives in their own right and can be used as to generate all intermediary subdivision steps (wedges, hulls, etc.) by properly setting their parameters shown in Figure 4.9. So, to yield a sphere the sphere section is set to $\alpha = 360°$, $\beta = 360°$, and $r =$ the radius of the resulting sphere.

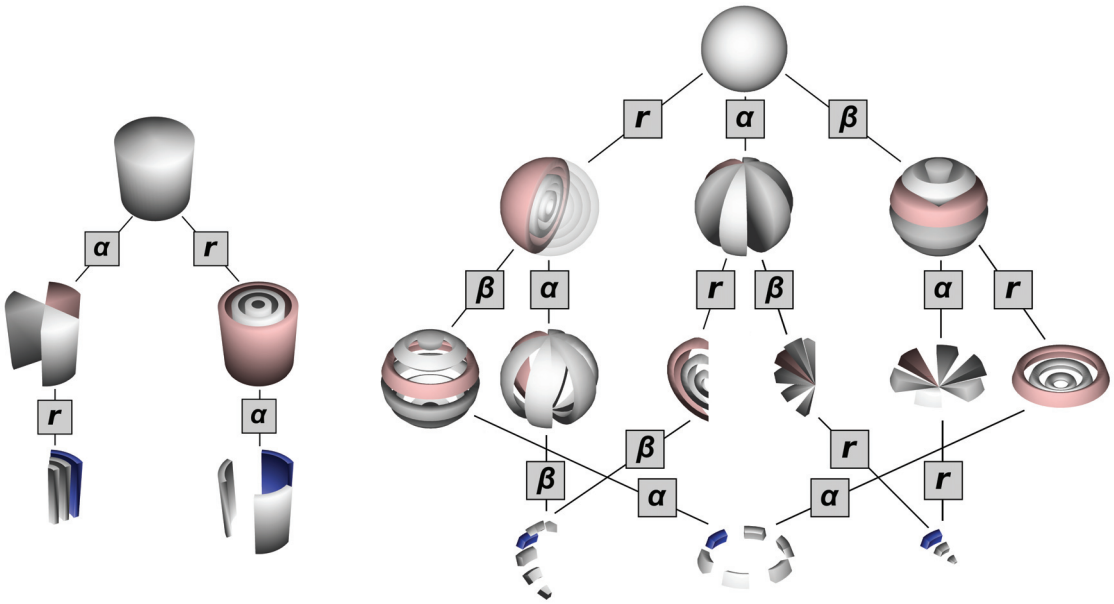


Figure 4.8: All possible polar cuttings of a cylinder (a) and a sphere (b). The red colored sections are cut in the next step. The blue colored sections are the resulting least elements – the building blocks of all the shapes above them.

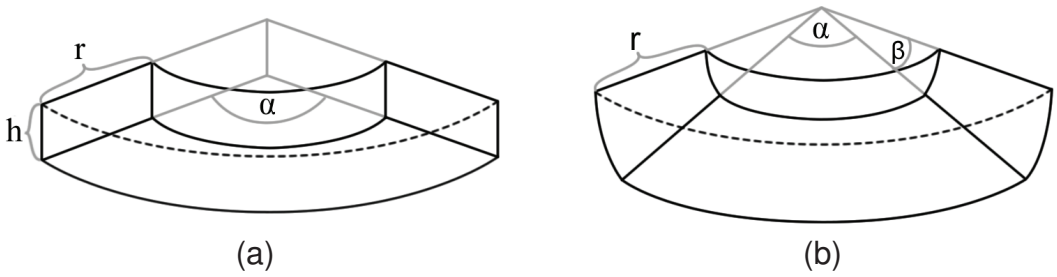


Figure 4.9: The cylinder section (a) and the sphere section (b) with their respective parameters.

3D Polar Treemaps / Radial Treecube

This technique uses the above introduced primitives sphere section and cylinder section for a subdivision layout together with an inclusion relation for edge representation. It uses the radial Slice-and-Dice layout that does not cut along the *x*-, *y*-, and *z*-axis, but along the two angles and the radius instead. It is combined with a sphere/sphere-section primitive yielding a new technique that can with equal justification be described either as a 3D Polar Treemap (being a 3D extension of Johnson's Polar Treemap [Joh93]) or as a Radial Treecube (being a radial variant of the Treecube technique [TON03]).

As there is more than one possible 3-dimensional extension of a circle (sphere, cylinder, cone, etc.), one can derive other variants as well – e.g. a cylindrical version. This is shown alongside the spherical one in Figure 4.10(a-c) together with the original 2-dimensional Polar Treemap. In the end, these visualizations are 3-dimensional subdivision techniques and share their strengths and weaknesses. Hence, 3D Polar Treemaps suffer from the same amount of occlusion as, e.g., Treecubes and cannot be recommended for deep hierarchies. On top of that, it may require additional interaction techniques like the cutting of regions or along planes as proposed for Treecubes in [TON04]. The advantage of the spherical variants compared to the Treecube is that no ordering among the nodes can be falsely inferred, as there is no front or back in a spherical primitive. The cylindrical variant forms a compromise between Treecube and spherical 3D Treemap as it still leaves one axis (top to bottom) for an ordering of the nodes.

Changing the inclusion relation into an adjacency relation leads to the extruded 3D Polar Treemap shown in Figure 4.13. This variant could likewise be called a radial cylindrical StepTree, if the squarified layout of the StepTree is altered into a polar coordinate Slice-and-Dice. It still has the Polar Treemap feel to it, but it is less occluded than the ones discussed here.

Spherical 3D Sunbursts

While in [SKW+07] a 3-dimensional version of the well known Sunburst technique has already been presented, it is based on a cylindrical extension of the circular Sunburst. The spherical 3D Sunburst is an extruded technique that makes use of adjacency to represent the parent-child relation. It is an example of how the sphere section can be used as a primitive by itself and (correctly parametrized) stacked on top of others of its kind.

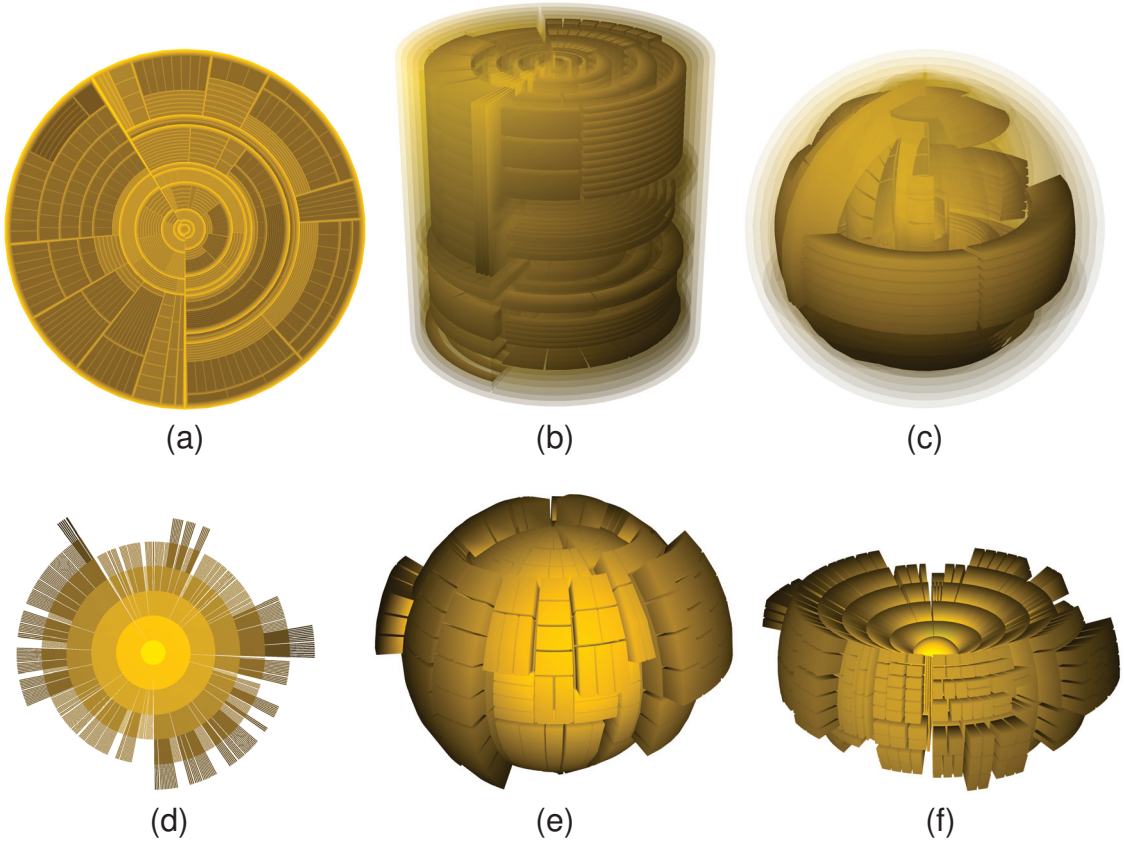


Figure 4.10: (a) The original 2D Polar Treemap (b) The 3D Polar Treemap variant “Cylinder” (c) The 3D Polar Treemap variant “Sphere” (d) The original 2D Sunburst (e) The 3D Sunburst variant “Sphere” (f) The 3D Sunburst variant “Wheel”.

Again, as there is more than one possible 3-dimensional extension for circles, the so far missing spherical 3D Sunburst is introduced in two variants. While both variants use a sphere and certain spherical cross sections as primitives, one allows the entire root to be covered (variant “Sphere”), whereas the other only allows to attach primitives to a narrow, horizontal belt (variant “Wheel”) – in part mimicking the original 2D layout. Both are shown in Figure 4.10(d-f) alongside the original 2D Sunburst. 3D Sunbursts tend to accentuate the leaves of a hierarchy, while occluding most of their interior nodes – quite alike to the original 2D Treemap layout without borders. At the same time, it is a very space saving way of displaying a rather wide trees that are not too deep. The variant “Wheel” lessens this effect a little by allowing viewers to peek inside the visualization from above and below. Yet, to do so it sacrifices some of the space that the variant “Sphere” uses so efficiently.

Spiral Steptree

At the core of this new technique stands a new primitive, termed a “leaning box”. This is a cuboid with its top being cut off at a certain angle. Stacking a number of them on

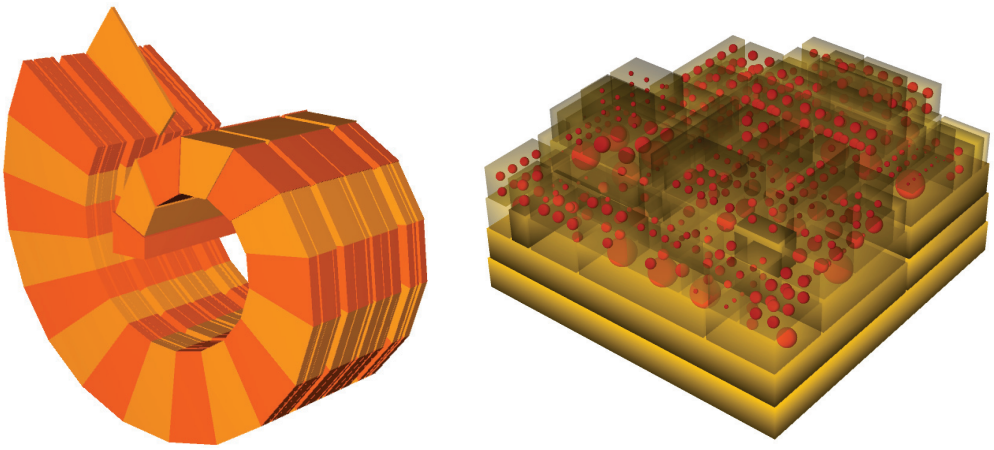


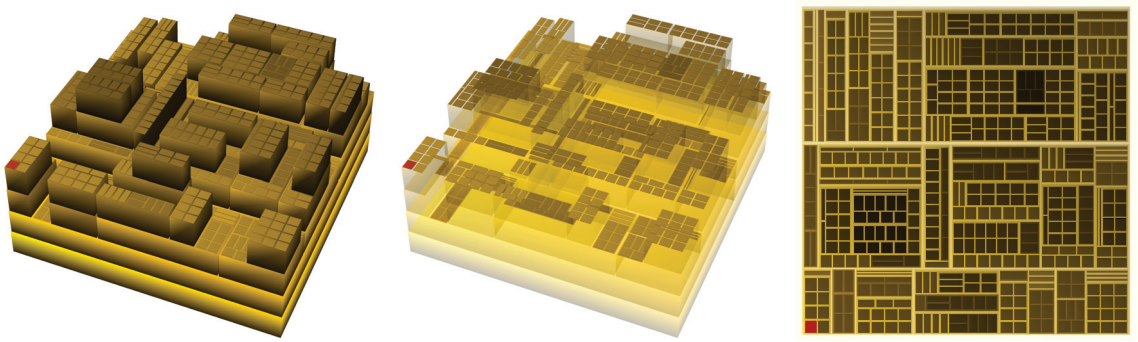
Figure 4.11: (a) Spiral Steptree that generates dense visualizations of hierarchies. (b) An example for a mixed adjacency/inclusion representation.

top of each other in a Steptree manner results in them leaning towards one side of the representation. If the tree is deep enough, such a visualization starts to curl around itself, forming a Spiral Steptree as shown in Figure 4.11(a). This proves to be an extremely space efficient way to display very deep, but narrow hierarchies. How tightly the representation curls up can be specified by the cut-off angle of the leaning boxes.

As illustrated in Figure 4.5(c), it is crucial to use layout methods that work well with a certain kind of primitive. The chosen layout subdivides the available space only along one dimension. This results in something that can be perceived as an Icicle Plot where the branches are not shown side by side, but are instead stacked behind one another. Because it is so compact, this technique is of no use to inspect individual nodes. Instead, it provides an overview for deep trees with expected uniform height. Any deviations from the uniform height will be instantly visible: shorter branches will show as gaps in the spiral, longer branches will peek out of the end of the spiral. To easily determine height differences between branches, an alternating color scheme has been chosen.

4.3.2 Mixing Different Node Primitives

The above means of combining different design possibilities along the four design dimensions to novel techniques already allow to obtain numerous hierarchical representations. Mixing two or more of them together into new techniques extends the possibilities to design an adjusted technique for a given scenario even further. It is interesting that mixing is not used much besides the few examples given in Chapter 4.1.2, despite its simple concept and its unobtrusive way to accentuate certain nodes and subtrees within the shown hierarchy. On top of that, it also allows to use specifically tailored design choices depending on the characteristics of a subtree. A first example was already shown in Figure 4.7, where spheres and open boxes where mixed to form a novel visualization that resembles a boxed collection of objects, which is not an uncommon use case [TON04].



(a) (b) (c)

Figure 4.12: These three visualizations show the same hierarchy with the leaf in the lower left corner being rendered in red to aid in comparing them: (a) mixed 2D/3D representation with the branches rendered in 3D and the leaves laid out in 2D on top of the 3D representation, (b) accentuating the leaves of the 2D Treemap by rendering the 3-dimensional primitives semi-transparent, (c) bird's eye view from above hiding the 3D part of the representation, leaving a 2-dimensional impression.

The first example mixes node primitives and parent-child relationships at the same time as shown in Figure 4.11(b). Here, adjacency is used to stack non-leaf nodes onto one another and inclusion to nest the leaves (shown as red spheres) into their parents. The level of transparency of the non-leaf nodes can be adjusted to emphasize either the overall structure (if made opaque) or give an impression of the leaf-level of the tree (if made transparent).

The second example in Figure 4.12 depicts a mixed Treemap with 2D and 3D node representations: (a) shows a mixed 2D/3D tree where all non-leaves are mapped onto stacked boxes in a Steptree-fashion, and all leaves are displayed as 2D Treemap on top of the boxes. (b) emphasizes the leaves, by adding transparency to the 3-dimensional boxes, leaving only the 2D-Treemap formed by the leaves clearly visible, but conveying their depth within the tree by the different heights of the 2D patches. (c) shows a bird's eye view, orthogonal to the top-faces of the 3D boxes – effectively yielding a 2-dimensional Treemap view of the mixed representation.

4.3.3 Parametrizing Node Primitives According to Derived Measures

Apart from the design strategies previously discussed, it is always possible to combine analytical means with visual cues defined by the design space. The following examples illustrate how such analytical measures can be used to encode additional structural information as properties of the node primitives, which is in this case an extruded 3D Polar Treemap.

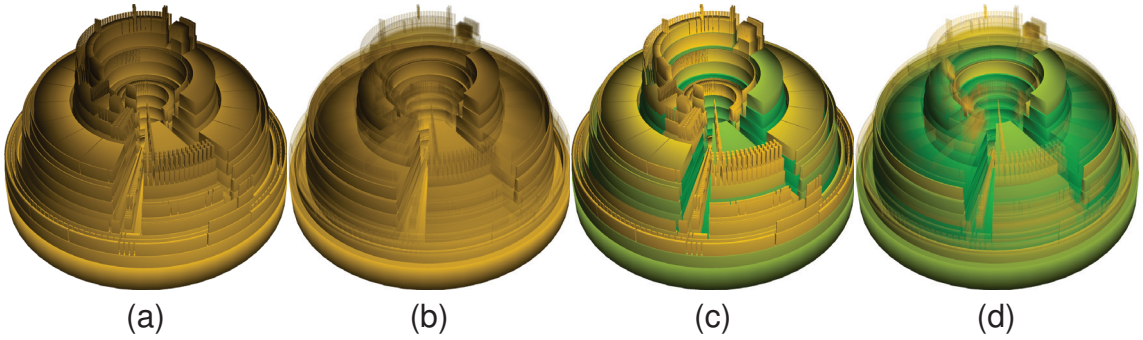


Figure 4.13: (a) A plain extruded 3D Polar Treemap, which is basically a Steptree-like technique, but uses the Polar Treemap subdivision layout (b) All nodes in the 3D Polar Treemap having a Strahler number smaller than a certain threshold are faded out (c) The eccentricity value mapped onto the color with green being mapped on smaller values, showing that the root is not the graph theoretical center of the hierarchy (d) A combination of (b) and (c)

Implicit Tree Skeletons

So far, tree skeletons have been investigated for node-link-visualizations only. They are defined as "the set of nodes and edges that are determined to be significant by a given metric" [HMM+99] and usually encoded in visual attributes of the edges. While implicit visualizations lack the edges, the idea of showing just an overview of the most important main branches of a hierarchy is equally intriguing for implicit techniques. What the main branches are and how far down to display them is usually determined by a branching complexity measure like the Strahler numbers [ADD+04]. An example for a 3D realization of an implicit tree skeleton using Strahler numbers is shown in Figure 4.13(b). Here, only nodes having a Strahler number greater than a certain cutoff value are shown opaque, all others are rendered semi-transparent.

Eccentricity Values

Eccentricity quantifies the distance between a node and the tree's graph theoretical center [RP89]. It plays an important role in rebalancing search trees, and in Operations Research where a negative correlation exists between eccentricity and cost efficiency. Often it is desired that the root of a tree is also its most central node. So, mapping the eccentricity value onto a tree representation reveals much about its balancing. Figure 4.13(c) depicts such an unbalanced hierarchy in which the root is not the most central node.

As the design space allows to map eccentricity and the tree skeleton to orthogonal visual cues, it is now possible to bring both of them together in one visualization. Figure 4.13(d) shows this combination. It gives an overview of the main branches and their balancing without being occluded by the large number of leaves.

4.4 Summary

This chapter and the design space presented herein set out to raise awareness for the problems involved in selecting and parametrizing visualization techniques to yield suitable representations. For the example of implicit tree visualizations, the different options to achieve such a representation have been unraveled and discussed. The definition of a conceptually complete and consistent design space and its practical applicability stand as a basis for a holistic classification and a rapid prototyping of known and yet unknown visualization techniques in this class. The systematic treatment of the entire design space allows the sporadic visualization user to get acquainted with the general design concept behind a whole class of visualizations, putting the known examples into a broader context. Taking this step back and looking at the big picture also makes it possible to break down visualization techniques into the recurring visualization design principles that they are made of. It is this externalization of implicit visualization design knowledge that provides a more generative perspective on this class of visualizations and by that a grip on its rather vast design space by means of rapid prototyping. A concrete software realization of the abstract design space has further been proven to be a useful tool for its hands-on exploration and the (combined) utilization of different design strategies to derive customized visualization techniques.

After this first step which looked closely at the representation level for a certain category of visualizations, the following chapter deals with the ties and dependencies between data and representation. It shows for two examples, how explorative graph visualization must address the two major characteristics of the data: its size and its graph type. Under consideration of large graph sizes, it is most important to devise efficient graph layouts that scale up well, while at the same time still faithfully communicating the features of the graph type. And the inverse is equally true: under consideration of the graph type and its expressive representation, it must still be ensured that it copes with graphs as large as they commonly occur in different applications.

Chapter 5

Visualization Solutions for Special Graph Classes

The discussion of the previous chapter gave an example of how complex the interwoven design considerations already are for just a subset of graph representations – but also for what can be gained by mastering them. Thereby, the importance of being aware of the different options governing the representational level of graph visualization techniques has been emphasized. Yet, to make the step from a well sought out parameter combination in a design space to a full-fledged graph visualization, the underlying data characteristics cannot be neglected. On the contrary, as the problem discussion has shown, the characteristics of the graph to be represented must in particular be observed to achieve an expressive, efficient, and effective visualization. These considerations about the data and also about the interaction with the data through the visualization go well beyond anything that can be captured by a single design space.

This chapter sets out to discuss the consequences of the data aspects on the representation – especially graph size and graph class. Both have considerable influence on a representation, and its runtime as well as its space efficiency. The larger a graph, the harder it is to compute a layout in reasonable runtime and with limited drawing space, because of usually polynomial computational complexity and quadratic growth of the number of possible edges to display with size. On the other hand, the simpler the nature of a graph class, the faster and the more space saving the layout can be generated by exploiting its properties. Hence, e.g., tree layouts can easily be achieved in linear time and in a space-filling manner – something hardly possible for general networks.

Besides runtime/space efficiency, graph size also influences the ability of fulfilling the (aesthetic) layout constraints posed by a specific graph class. This can be found throughout the visualization literature, e.g., for certain representations of bipartite graphs which tend to generate a cluttered appearance for larger node sets [IMT09] and which were thus extended to 3D to lessen this effect.

The design consideration it takes to construct representations that are specifically adapted to graphs of large sizes and/or to graphs of a certain class are exemplary discussed in this chapter. As there are a multitude of possible combinations of different graph sizes and graph classes, two representative examples are considered:

Trees with up to a million nodes as an example for the visualization of large graphs. An efficient overview visualization[1] is proposed, which balances space utilization with communicating properties of the tree. While there exist many tree visualizations, only a very few can cope with trees of this size. Hence, presenting a novel representation approach for this graph class/size-combination significantly furthers the visualization options for this case.

Bipartite graphs with thousands of nodes as an example for the visualization of a specific graph class. A (re-)orderable visualization[2] is introduced, which allows attribute-based as well as topology-based tasks on attribute-rich bipartite graphs. As the visualization of bipartite graphs is in general underdeveloped in contrast to the wide-spread occurrence of this class, the presented visualization technique is a useful contribution to this field.

In contrast to the last chapter, both visualization techniques are explicit, node-link representations. Beyond the pure layout, interaction techniques that tie in with the respective representations are discussed in detail, as pure visual inspection without the possibility of interactive inquiries does not yet make an explorative analysis. Furthermore, the strengths of both techniques are exemplified by exploring different modifications and enhancements, as well as by applying them to different scenarios and data sets.

5.1 A Point-based Visualization for Large Trees

Large trees occur in many fields, e.g., cluster dendrograms of gene expressions in life sciences or ontological hierarchies in information sciences. They rarely have more than a million nodes and as they are stemming from real-world applications their overall structure is usually not regular or even balanced. Efficiently and yet faithfully representing such a large tree and communicating its overall characteristics (e.g., imbalances) is a challenge that not many existing visualization techniques can face.

Efficiency in graph layout has long been understood only in terms of the needed runtime [BETT99]. Besides CPU time, other limited resources like the available drawing space have been only implicitly taken into account by minimizing the area of the drawing as a desirable aesthetic constraint. Yet, this has changed dramatically over the last years, as the performance of consumer CPUs has exponentially increased, as well as the size of the graphs/trees to be drawn – whereas the available screen resolution has grown much slower. Hence, the concept of *space-efficiency* receives more and more attention these days, trying to make the most out of the few pixels available. Space-efficiency is closely related to the *space-filling property*, which is a desirable condition for space-efficient hierarchical layouts [MR10].

[1] appeared as "Point-Based Tree Representation: A new Approach for Large Hierarchies" in the Proceedings of the IEEE Pacific Visualization Symposium 2009, extended version "Point-Based Visualization for Large Hierarchies" accepted with minor revisions for the IEEE Transactions on Visualization and Computer Graphics

[2] appeared as "Visual Analysis of Bipartite Biological Networks" in the Proceedings of the Eurographics Workshop on Visual Computing for Biomedicine 2008

Hence, the question is how to sensibly fit a tree with up to a million nodes into the available drawing space without losing the expressiveness and effectiveness of the representation? While space-efficiency poses the main challenge for depicting a tree with that many nodes, after all it should also still be recognizable as a tree, and its features (balancing, depth, width, etc.) should be discernable. The space-filling property as exhibited by some tree layout techniques, such as the Treemap and its successors, is an acknowledged design constraint to achieve such a visualization. For example, Treemaps expressively and effectively depict trees with up to millions of nodes [FP02]. They are space-filling by design, as they utilize the available screen space entirely by subdividing the space itself into a representation of the given tree. They use nested shapes instead of nodes and links to represent the hierarchical structure and scale well above all known node-link representations. This is due to the fact that they utilize the available screen space entirely by molding the space itself into a representation of the given tree. This is done either through subdivision by recursively carving the drawing area out of the available space for each subtree, or through packing by arranging the subspaces representing the subtrees in the available space.

While this approach naturally yields a space-filling tree visualization, it is not applicable to node-link-representations. These rather see the tree with its nodes and edges as the object that needs to be shaped to fit the available space. Many node-link techniques try to optimize their usage of the available screen space by either packing the nodes as tightly as possible to minimize the used space or by distributing the nodes as evenly as possible to make best use of the available space. Yet, both approaches prioritize the optimal usage of space over the depiction of a tree's characteristic features, which results in a loss of effectiveness. On top of that, tight packing as well as evenly distributing destroy the hierarchical impression a tree visualization should convey. Hence, in contrast to implicit representations they need to draw the edges as the parent-child relationship cannot be discerned from the layout alone.

This chapter sets out to introduce a more balanced layout approach. The presented layout allows to distinguish between dense and sparse subtrees while still making good use of the space. This is achieved by combining the most space-efficient visual representation of a node – the single point – with a sophisticated, yet computationally fast hierarchical placement scheme, consequently adapted from the point-based rendering paradigm. The *point-based tree visualization* technique based on this layout scheme works in such a way that it produces continuous areas for dense subtrees in a space-filling manner and dissociates into a regular node-link appearance for sparse subtrees. If the tree is unbalanced or partially very narrow, it even leaves parts of the screen space empty – yet not unused, as these parts serve to communicate the said imbalances and sparser subtrees. This is exactly what distinguishes it from other approaches which look more or less alike, because the tree is not allowed to exhibit its characteristics, but instead squeezed into or spread out across the available space. The tree layout resulting from this approach has several advantages:

in terms of efficiency – it is space- and runtime-efficient, as it is provably space-filling and runs in $\mathcal{O}(n \log n)$, and can thus cope with trees containing up to a million nodes.

in terms of expressiveness – as an overview visualization, it truthfully shows the overall features of a tree's structure or node attributes.

in terms of effectiveness – it provides a fixed placement strategy that aids in orientation and comparison, but is at the same time highly parametrizable and can be adapted to different tree characteristics and properties of the drawing space.

In the first part of this chapter, the layout is introduced as an overview visualization for which the features listed above make it a good fit. Especially runtime- and space-efficiency are two important properties of an overview that must be generated fast and potentially fit in a small area alongside a detailed visualization in an overview+detail combination. A threefold comparison between the point-based tree visualization and existing space-filling approaches is given in the second part of this chapter. Firstly, it relates the point-based layout conceptually to other space-filling approaches to check if it is indeed space-filling. Secondly, it investigates under which conditions the different layouts become space-filling. And thirdly, it determines how well the space is filled in terms of actual screen utilization (numerical evaluation) and in terms of its readability (preliminary user study) in order to substantiate the claim of being a good compromise between space usage and discernability of a tree's characteristic features. A third part finally illustrates the parameter space of the point-based visualization and exemplifies how changes on data and representation level can be addressed by altering and extending the basic layout method.

5.1.1 A Point-Based Tree Layout

When large hierarchies need to be displayed, visualization designers look for techniques that make the best use of the available screen space. This is where space-filling layout techniques come into play. Yet, in the past, space-filling techniques have often been set equal to implicit tree layouts. It seemed that only implicit techniques with their 2-dimensional graphics primitives were able to fully fill the available screen space. Hence, explicit techniques, which try to maximize their usage of the screen space, usually call themselves "space-optimized" or "space-efficient." The layout presented here is a first attempt to achieve an explicit space-filling layout in the sense of the following definition:

Definition: A tree visualization is called *space-filling*, iff

$$\text{Ink-Paper-Ratio} = \frac{|\text{used pixels}|}{|\text{available pixels}|} = 1$$

This condition formulates the usual understanding of the term "space-filling", namely that every available pixel is used. The Ink-Paper-Ratio [FT94] is the quotient of Tufte's Data Density and Data-Ink-Ratio [Tuf01]:

$$\text{Data Density} = \frac{|\text{nodes to display}|}{|\text{available pixels}|} \qquad \text{Data-Ink-Ratio} = \frac{|\text{nodes to display}|}{|\text{used pixels}|}$$

To make sure that the above condition is an inherent property of the layout and not an unintended side effect, the space-filling property holds only under the condition that it is not achieved by

(a) using a screen space that is much too small for the hierarchy to be laid out, resulting in a space-filling representation due to massive overplotting, or

(b) arbitrarily blowing up individual nodes to occupy remaining whitespace, just to utilize the full screen and become space-filling.

Inspiration

The idea for the proposed layout method stems from the area of point-based graphics [GP07]. By point-based methods, triangular graphics primitives, of which 3-dimensional renderings mostly consist, are replaced by point primitives. This makes sense, as today's high-resolution models consist of millions of triangles, which cover only a relatively small screen area and often share the same pixels. This results in a computational overhead, which is usually not justified by the gained result improvements. Especially so, as the same result can be computed with less effort by using point primitives instead, which are easier to render. By a careful arrangement, even surfaces can be represented using nothing but points. Hereby, techniques like the $\sqrt{5}$-sampling [SD01] make sure that indeed a closed surface will be shown while at the same time minimizing the number of points used to do so. The left side of Figure 5.1 shows an example of a simple regular point placement leaving gaps in steep regions of the surface, whereas the right side is generated by the more advanced $\sqrt{5}$-sampling and achieves complete coverage.

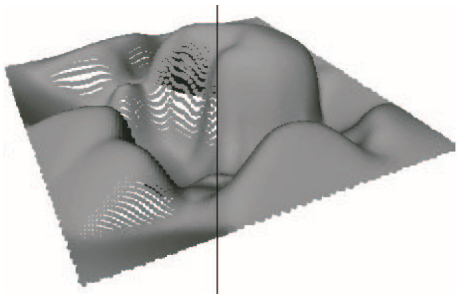


Figure 5.1: A point-based rendering of a surface with a simple point placement technique (left) and the placement according to the $\sqrt{5}$-sampling (right) [SD01].

Hence, the point-based rendering paradigm does not only make use of the most space-efficient graphics primitive, the point, but also incorporates methods to place points in a space-filling way. Both aspects are highly desirable for a visualization of large data sets as well, as there is no other primitive of which one could fit more onto the available screen space – especially if it can be done in a space-filling manner. So, even though the point-based graphics originates from a different context, it can be utilized for a space-filling position of a tree's nodes.

Layout Technique

Arranging as many points as possible in the form of a sequence or matrix as pixel-based visualization techniques do is one thing, expressing a hierarchical relationship between

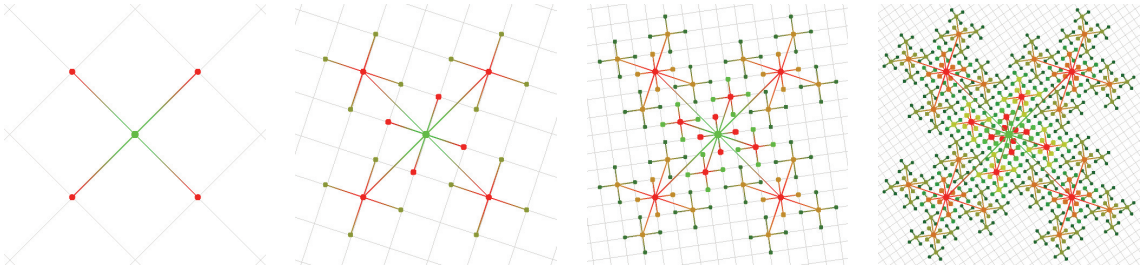


Figure 5.2: Four recursion steps of the $\sqrt{5}$-sampling method.

points just by their arrangement a completely different one. The benefit of a hierarchical placement strategy would be that edges are no longer needed to represent the parent-child relationship. The parent of a node can be found just by their unambiguous positioning with respect to each other, very much like implicit tree visualizations do. The benefit is that the omitted edges result in less visual clutter (locally very dense regions) and hence more nodes being visible.

Interestingly, the $\sqrt{5}$-sampling mentioned above uses a hierarchical sample scheme for positioning points at possibly undersampled, blank spots around other points. At each step of this technique, the starting grid is refined by a rotation of $\arcsin(\sqrt{5}) \approx 27°$ and a reduction of $1/\sqrt{5}$ of the distance between two adjacent grid points. Around every undersampled point, four new points are inserted at the nearest position in the current grid. Figure 5.2 illustrates the steps of this algorithm. This shows how the overall density increases with every recursion step and how gaps are filled in between the points.

In Figure 5.2, additional lines where included that are not part of the original $\sqrt{5}$-sampling method. These lines (edges) between the points (nodes) already hint at a possibility to map a tree structure onto the resulting point positions. While the $\sqrt{5}$-sampling is originally adapted according to surface properties, the proposed point-based layout technique uses it to adapt to tree characteristics. This is done in the same recursive, step-wise manner as the $\sqrt{5}$-sampling method itself:

(a) Starting with the root in the center of the screen, the root's first 4 children will be arranged around it.

(b) In the next step, the next 4 children of the root and the first 4 children of the previously laid out nodes are positioned by a rotation and scaling according to the $\sqrt{5}$-sampling.

(c) Then, the same procedure is repeated to layout the next 4 children of the root node, the next 4 children of the nodes laid out in step (a), and the first 4 children of the nodes that have been added in step (b).

(d) This last step of Figure 5.2 adds another 4 children around the root node, as well as 4 children to all nodes from steps (a)-(c).

Algorithm 1 The basic layout algorithm.

```
1:  θ ← arcsin(5^−2)                                   ▷ constant rotation angle ≈ 27°
2:  procedure LAYOUT(R,P,L,α)
                                                        ▷ R = the (sub)tree's root
                                                        ▷ P = the position of the root
                                                        ▷ L = distance between root and first 4 children
                                                        ▷ α = directional angle of first child
3:      if L ≥ 1 then
4:          children[] ← getChildren(R)
5:          sort(children[])
6:          for i ← 1 to getSize(children[])) do
7:              level ← ⌊i/4⌋
8:              position ← (i − 1) mod 4
9:              L′ ← L/(5^(level/2))
10:             α′ ← α + θ ∗ level + π/2 ∗ position
11:             P′ ← (P_x + L′ ∗ cos(α′), P_y + L′ ∗ sin(α′))
12:             LAYOUT(children[i],P′,L′ ∗ 5^−2,α′ + θ)
13:         end for
14:     end if
15:     drawPoint(P)
16: end procedure
```

This procedure is repeated until all nodes of the tree are positioned. More formally, the procedure is stated in Algorithm 1.

The conditional clause in Line 3 of the algorithm is not actually necessary, but it speeds up the layout by preventing the algorithm from needless overplotting once it gets into subpixel ranges. Another addition to the layout is Line 5, in which the children are sorted – optimally by the size of their rooted subtrees. This ensures that the largest subtrees will be positioned on the largest available drawing areas in order to use the space as best as possible. The remainder of the algorithm is exactly the procedure described above, with Lines 9 and 10 realizing the scaling and rotation, Line 11 computing the actual coordinates of the current child, and Line 12 making the recursive call to layout the subtree rooted in this child node. The recursion breaks off, if a node is a leaf or the distance between parent and child is less than 1 pixel. Because of the sorting step, the layout runs in $\mathcal{O}(n \log n)$ with n being the number of nodes in the tree. If the sorting step is carried out once and for all in a preprocess, the pure layout without the sorting even runs in $\mathcal{O}(n)$. The result of this basic layout algorithm is illustrated in Figure 5.3(a) for the DMOZ hierarchy. This hierarchy is used as an example tree for most figures in this chapter. It is the categorization hierarchy of the Open Directory Project dmoz.org with 765,328 nodes and 585,217 leaves thereof (snapshot from 01-JUN-2009). This categorization is used throughout the internet for classifying websites. In this particular snapshot of the data set 4,597,967 websites are associated with at least one of the classification's categories.

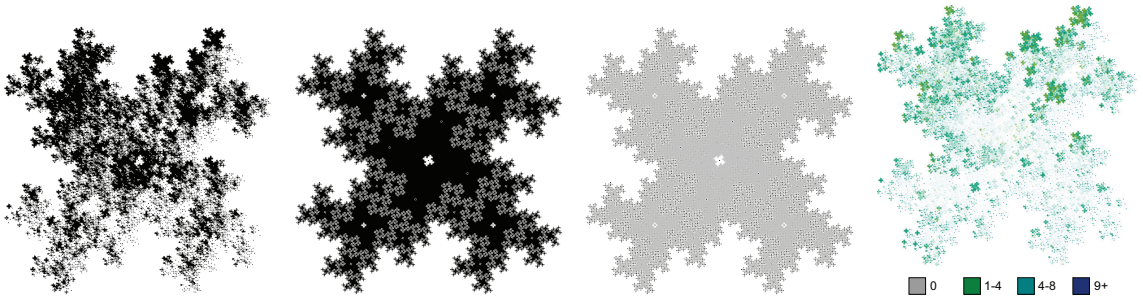


Figure 5.3: (a) The basic layout for the DMOZ hierarchy. (b) A regular tree with a uniform width of 16 and a depth of 6 displayed with the same basic layout. (c) The same tree as in (b) but this time rendered using the lightness adjusted layout with content-aware scaling. (d) The DMOZ hierarchy color coded by subtree depth and lightness adjusted.

Coloring Techniques

The technique uses color to enable the perception of structure and attributes, as well as to enhance comparability between subtrees of different sizes. The latter is probably the most important, as it addresses a fundamental issue of the layout: as not all siblings can be assigned the same space, subtrees placed later, because their root node is further down the sorted node list, may appear more tightly packed and thus stand out. This can be seen in the center of Figure 5.3(a), where subtrees seem to be heavily weighted and at least as important as the ones mapped on the outer positions – even though it is not the case.

As a solution to this issue, a small adjustment to the basic layout algorithm produces the exact same node positions as the basic layout, but changes the lightness (in HSL color space) of the points, so that they visually appear evenly distributed in case they are. This is achieved by laying out all subtrees into a buffer of the same size as the largest available drawing area for a subtree and then to iteratively scale that buffer down by $1/\sqrt{5}$. The scaling is done in a content-aware manner, so that each pixel in the resulting smaller buffer is set to the average lightness of the buffer patch that it subsumes. This is done until the final layout size for a subtree is reached. Conceptually, this process is nothing else than a run of the basic layout algorithm without Line 3 and then, after everything is laid out, a "backwards" run of the same algorithm. In this second, bottom-up run, it now collects all drawn and blank (white) pixels of a certain layout step, locally aggregating their as well as their parent's lightness values and setting this aggregated value as the new lightness of the parent. This is repeated, until the desired smaller resolution is reached. The result is shown in Figure 5.3 for a regular tree, where Figure 5.3(b) depicts the basic layout and Figure 5.3(c) the lightness adjusted by content-aware scaling. As it can be seen, the basic layout exhibits the said problem of subtrees standing out without them being actually more important or dense than any of the other subtrees on the same level. As the tree is regular, this problem cannot be solved by sorting the subtrees according to their weight. In contrast, the adjusted layout shows an equal point distribution everywhere, clearly communicating the high structural regularity.

Enhancing the perception of structure is important for certain exploration tasks that involve the width (max. number of siblings) and the depth of the hierarchy. The point-based layout indicates by design where large subtrees are located by generating densely packed regions/spots. This allows to make informed guesses about balancing issues of the hierarchy, but not whether a subtree of notable width or depth is the cause of a dense region. In order to clarify this, a color coding of a subtree's depth or width onto its nodes is proposed, so that the cause of a dense region becomes apparent. Figure 5.3(d) shows such an encoding of the tree's depth into colors for the DMOZ data set. From the color coding it can be seen that except for a few dense spots containing deep subtrees, the cause for the accumulation of nodes are mostly shallow and therefore rather wide subtrees. When the focus of exploration lies on numerical node attributes, a similar color coding can be used analogously, as shown in Figure 5.4(a).

In general, when using colors in addition to the lightness adjustment, this poses additional constraints on the color scale to be used. In order to not interfere with the adjustment, it is necessary to chose only colors of equal lightness. On top of that, the lightness adjustment interferes with the distinguishability between the colors. Therefore, it has proven to be more effective to use a discrete color scale with no more than 5 different colors. In the colored layouts of the DMOZ data set – e.g., in Figure 5.3(d) and 5.4(a), four colors were used for the depth and also for the number of categorized websites. The colors as well as the thresholds between the different discrete steps can be interactively changed to highlight a user-specified attribute range with a certain color.

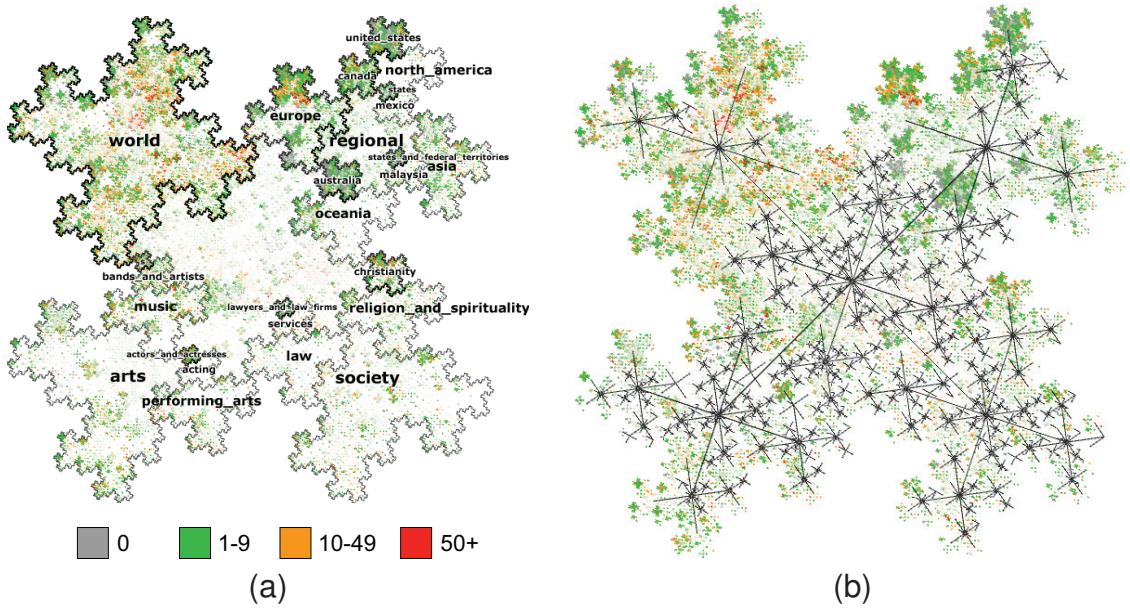


Figure 5.4: (a) Point-based visualization of the DMOZ hierarchy. It contains 765,328 nodes of which are 585,217 leaves. The node colors denote the number of websites in the corresponding category. Dark patches indicate larger subtrees in contrast to the lighter regions of smaller subtrees. (b) Adaptive edge representation: edges have been added in sparsely populated regions.

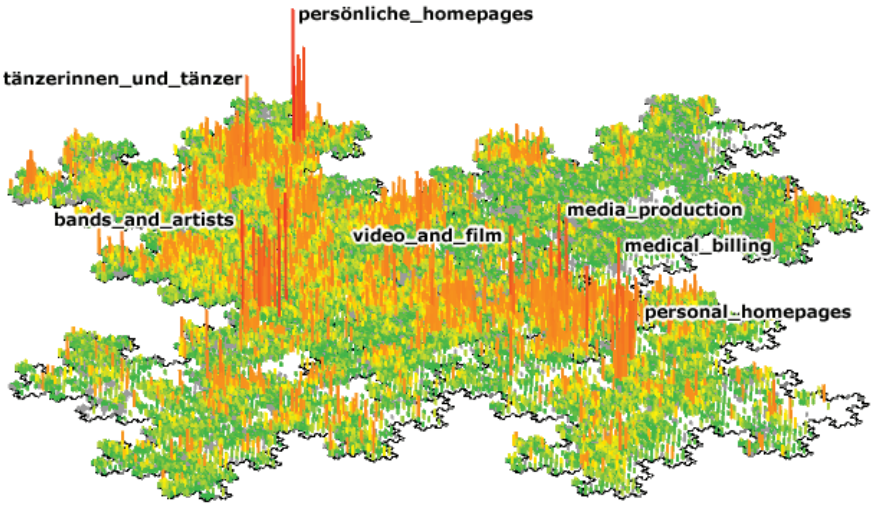


Figure 5.5: An example of the 3D extension for the number of websites in the DMOZ data set. As in this case negative values cannot occur, the bottom of this visualization does not show any peaks.

3-dimensional Extension

The coloring is constrained by a number of aspects as mentioned: one can use it either to communicate structural properties or numerical node attributes, and the utilized color scale should be discreet with colors of equal lightness values. To alleviate these problems, it is possible to employ the third dimension as another layout parameter besides color to map values on. A 3D extension can be used to display values more prominently and continuously in the form of bars or peaks rendered on top of the 2-dimensional layout. In this case, a numerical node attribute is mapped onto the height of these bars, so that high values stand out not only in terms of the coloring but also topologically. Negative values are rendered below the 2D layout, likewise achieving a good visual distinction for low values. An orthogonal projection is used for this 3D rendering to lessen distortion and to enhance the comparability between different peaks. Overall, this shifts the focus of the visualization towards the node attributes and displays the hierarchical structure merely as contextual information. It can be seen in Figure 5.5 for the DMOZ data set, that the 2-dimensional silhouette of the hierarchically arranged regions are mostly occluded by the 3-dimensional bars. This balance between showing attributes and structure can interactively be changed by simply tilting the representation more into an orthogonal view from above/below to see the structure or into a sideways view to compare attributes.

Interaction Techniques

The overview technique described above is designed in such a way that it communicates an overall impression of a tree's characteristics. Additionally, a basic set of interaction techniques is helpful to engage the user and to allow further drill-down exploration from the initial overview and thus to confirm first impressions gained from it. Each of the interaction techniques listed in the following is exemplified in Figure 5.6.

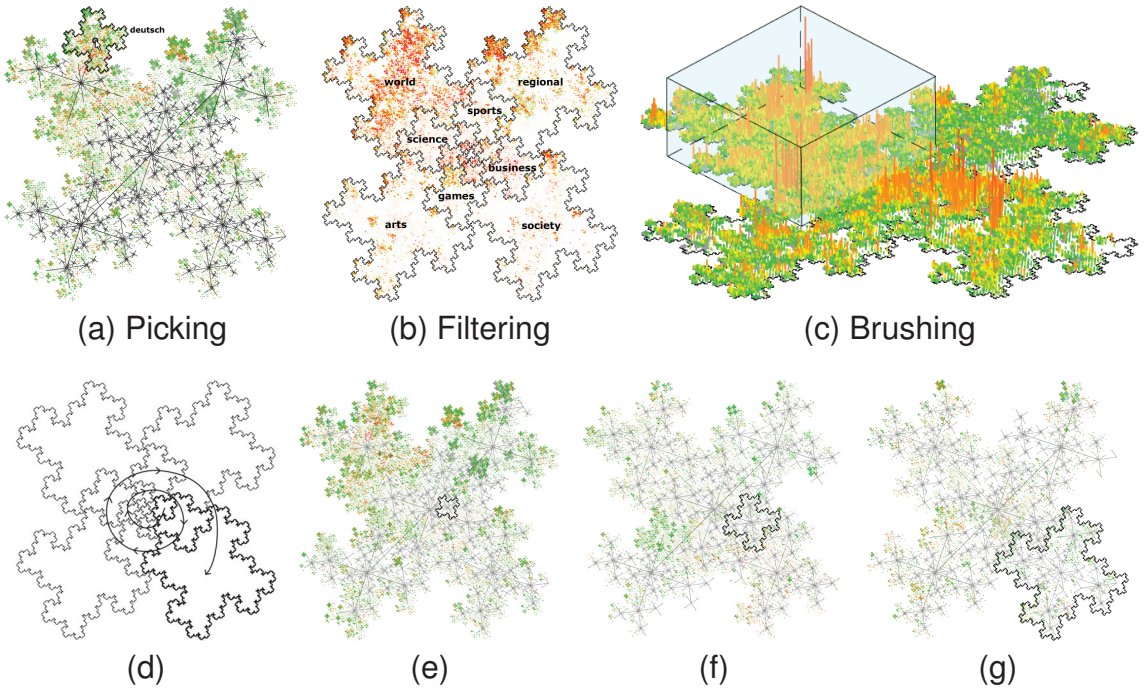


Figure 5.6: Basic interaction techniques: (a) Picking the node/subtree “deutsch” in the DMOZ hierarchy. (b) Filtering all nodes that have more than 10 websites and that lie deeper than 7 levels below the root. (c) Brushing the region around the highest attribute values from Figure 5.5. (e)-(g) The subtree “adult” is enlarged by an outwards rotation along the lines of the rotation scheme shown in (d) onto one of the larger areas for closer inspection.

Picking a certain node and by that also the part of the hierarchy that forms the entire subtree rooted in this node can become quite painstaking for a small point feature in a dense layout. To counter these difficulties and aid in picking, a hovering effect is introduced that does not only show the label of the node currently under the mouse cursor, but also outlines the region of this node’s subtree. Once the desired node/subtree is found, a mouse click directly selects the node.

Filtering out nodes is suitable to quickly select a subset of nodes from a value-range point of view. Range sliders can be used to specify width or depth ranges of interest and it is also possible to use different filters in conjunction. Analogous to the selection by structural properties, filtering can equally be driven by numerical node attributes, e.g., the number of websites in the DMOZ example. Hence, it can be used to specify nodes using structure and attributes, likewise.

Brushing a certain region of the visualization and thereby selecting attribute values within it is a third possibility. A resizable, rectangular brush is provided, which extends to a cuboid for the 3D version of the layout. This allows to select attribute values that lie in close proximity to one another, but do not necessarily belong to the same subtree.

Zooming, including rotational zooming (schematically depicted in Figure 5.6(d-g)) and geometrical zooming is important for a more in-depth exploration of the overview. Zooming is especially useful as it is an inherent property of the layout that nodes on the same level do not necessarily occupy the same amount of space. Hence, an enlargement of smaller regions is an essential interaction.

Visual Cues

Additional visual cues are necessary, to enrich the visualization with elements otherwise not shown, as they are not part of the basic layout: edges, paths, subtrees, and dense, potentially overplotted regions of the layout.

Edges. Omitting the edges in the representation is necessary, as they would occlude the tightly placed points which leave no space for drawing edges. Yet, edges are also necessary to connect otherwise loosely placed points and thus to relate them to the overall hierarchy. This is especially important, as the point-based layout is not layered in the sense of classical tree layouts which position nodes of the same depth on the same layer. Hence, an adaptive approach as presented here draws only unobtrusive edges and leaves out the rest. An edge is considered unobtrusive, if it does not cross any region with an average lightness below a given threshold. Hence, edges would run only across relatively light and thus sparsely populated regions and not obscure too much information. The threshold can be interactively raised or lowered, resulting in less or more drawn edges, respectively. The placement of edges is done top-down from the root to its children and so forth. If for any node the edge to its parent is not drawn, for it would cross a denser, darker region, the top-down traversal is skipped for the entire subtree below it. This prevents solitary edges from being drawn, because they tend to disturb the orderly layout and falsely suggest the existence of unconnected components. The result for the DMOZ data set is shown in Figure 5.4(b).

Paths. Besides edges, the display of entire paths may be needed for an exploration task. They can be overlaid on the layout on demand for selected nodes. When tilted with the mouse to get a 3D representation, the overlaid path is shown as a series of poles and a semi-transparent "wall" connecting them. The height of the poles are scaled to a numerical attribute – in this case to the number of categorized websites of the corresponding subtree. This can be seen in Figure 5.7, where the 2D and 3D versions are shown. Since one subtree is part of another, bigger one, the portion of the pole that is made up by the value of the picked node is colored in light blue instead of gray. This way, one can get an impression of how the individual subtrees contribute to the overall values. In the example from Figure 5.7, the "USA" subtree has some 600.000+ websites categorized in it. From the visualization it can be seen that this is quite a large portion of the overall number of about 4.6 million categorized websites.

Subtrees. Each subtree is laid out in a fractal shaped region. It is an intrinsic feature of the layout that these regions are packed tightly beside one another but do not overlap in order to fulfill the space-filling property. Yet, while this maximizes the usage of the available drawing area, it also crams the points in the visualization and makes it sometimes less obvious to see where a subtree's boundaries are. On the other hand, drawing all of the boundaries would occlude most of the placed pixels. Hence, a more balanced approach is

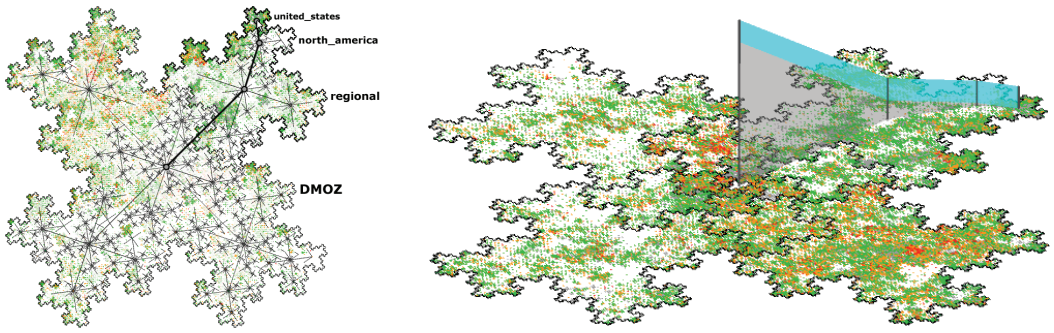


Figure 5.7: Overlaying the path from the subtree “USA” to the root node in 2D and 3D.

needed that emphasizes only some, preferably the most important subtrees. As the layout is tuned to visually bring out denser subtrees by compact point placement balanced by lightness adjustments, the average lightness value of a subtree is used to determine if a subtree should be drawn with or without boundaries: the larger a subtree is, the darker it appears and the more it sticks out, hence the more reasonable to draw its boundary to aid in clearly distinguishing it from its neighbors. Additionally, for each drawn boundary a label is placed that identifies the bounded subtree by its root’s name. To ensure fast and non-overlapping labels, the particle-based labeling algorithm from [LSC08] is used. It integrates quite naturally with the layout, as each positioned point can likewise be considered as a particle not to be occluded by the labels. Very light and barely recognizable points due to the lightness adaptation are excluded from the list of particles and thus available to be allocated by labels. The lightness threshold can interactively be changed to make boundaries and labels appear for more or less subtrees. An example for the DMOZ data set is depicted in Figure 5.4(a).

Dense Regions in the Layout. Despite all efforts to lessen overplotting (by a space-efficient arrangement) and its effects (by lightness adaptation), the layout cannot prevent overplotting from happening. Hence, it should at least be communicated to the user, so that it becomes clear where further exploration may be needed. This is addressed by providing a GraphSplat [vLdL03] as a linked view to indicate regions of overplotting in this heatmap-like visualization. This is a common concept when visualizing a large number of items [FP02], where cluttering artifacts and overplotting occur often and obscure the view on the actual number of data items. Here, the GraphSplat allows zooming and panning and is directly linked to the point-based layout. It can be used to scale a dense region of the layout to the size of the available screen space. Since zooming-in results in a sparser layout, the lightness values of the now zoomed-in regions are recomputed and more adaptive edges can be drawn as there are less points to be occluded in the same screen space. An example is shown in Figure 5.8, where a dense region of the original layout is identified in the GraphSplat visualization and then zoomed-in to view details of this region.

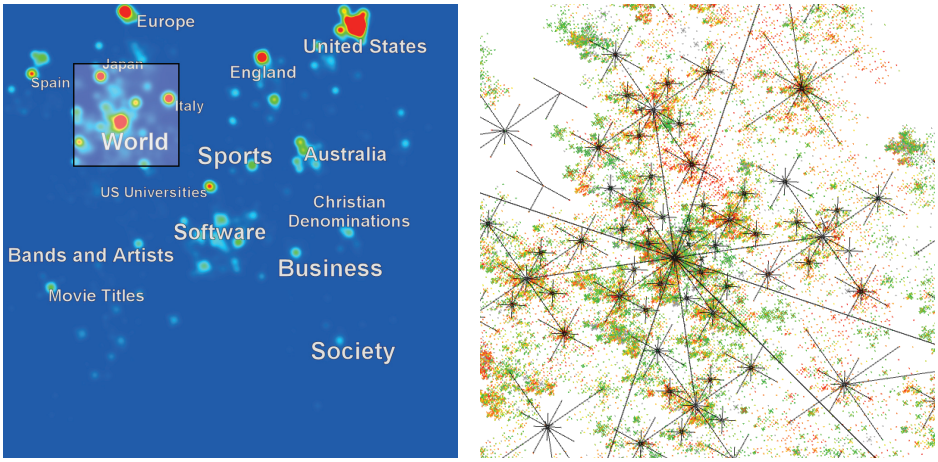


Figure 5.8: GraphSplat of the point-based layout from Figure 5.4(a) with a dense region being selected (a) and then zoomed-in in the linked hierarchy visualization (b).

Utilizing the Basic layout

The basic layout algorithm is used and integrated in the **JA**VA-based **M**ultipurpose **E**nvironment for **S**imulation **JAMES II**[3], an extendable simulation framework [HR09]. Within the graduate school "dIEM oSiRiS", this framework stands as the back-end behind most of the research – be it the development of modeling formalisms and tools, new simulation algorithms, or even e-learning strategies. The JAMES II framework is one of the first to support hierarchical model structures as they occur in multi-level models [UEJ+07], e.g., when combining agent-based and population-based approaches in one model. For validating and debugging purposes, these hierarchical models need to be visualized. As these hierarchies are large (from around a few 100.000 up to a million nodes) and very regular, the point-based layout is a perfect fit: irregularities (possible bugs in the model) stick out and are easy to perceive. On top of that, filtering by width can be used to identify submodels that have an unusually large or small number of components which in turn may hint at a faulty model composition. Figure 5.9 shows the integration of the point-based layout in the JAMES II framework depicting a MultiLevel-DEVS model with 6 hierarchy levels and about 300.000 nodes. Upon inspection of this model, the developer can identify and zoom in on irregularities, select the nodes in question, and investigate the problem in adjoint views.

All in all, the above considerations exemplify the intricacies of visualization design under consideration of data and layout constraints, interaction facilities, and added visual cues to support a multitude of possible exploration paths. But also, the example of the effective integration of the visualization into the simulation framework showed the benefits of going through all these considerations. The following second part of this chapter describes the conceptual ties of the point-based layout to other space-filling layout techniques and compares it to them.

[3] http://www.jamesii.org

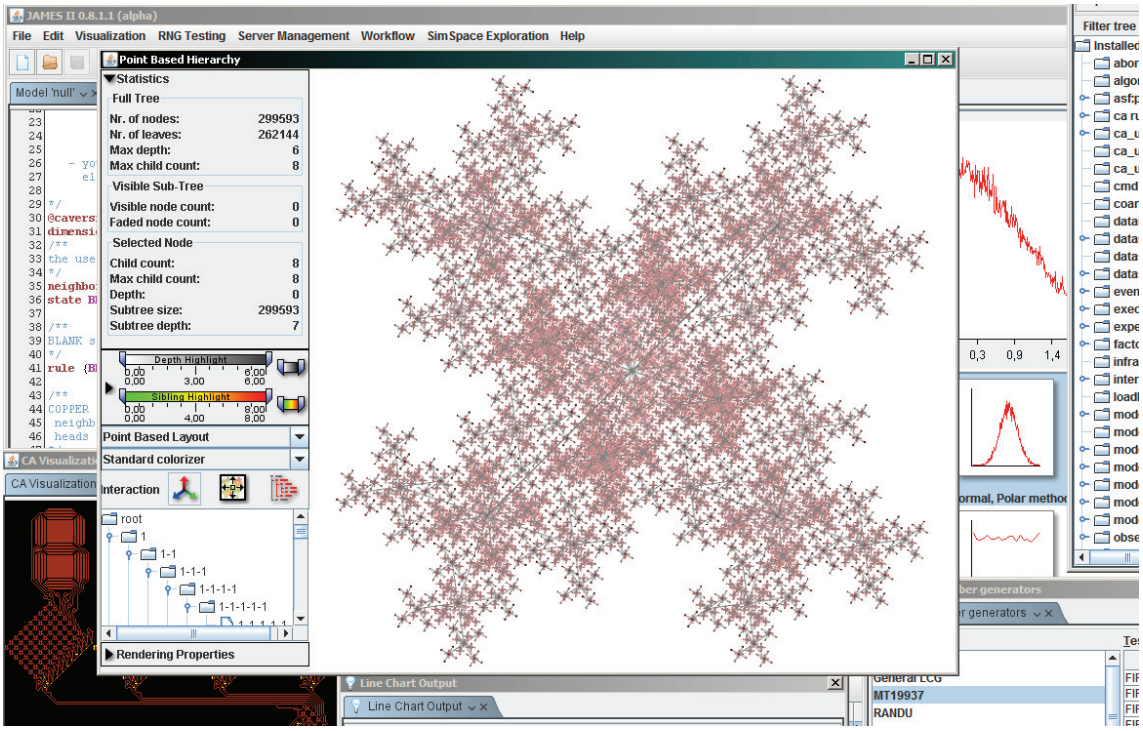


Figure 5.9: The JAMES II framework [HR09] showing a point-based layout of a hierarchical model with 300,000 nodes.

5.1.2 Putting the Point-based Tree Layout in Context

In order to assess if and how well the layout indeed fills the 2-dimensional drawing space and still remains readable, this part evaluates the layout and relates it to others. When relating the technique to other space-filling techniques, it can be seen that they have quite different characteristics, which divide their spectrum into several categories. This is not surprising, as the term "space-filling" appears with different notions in the literature.

Hence, to actually compare and relate different space-filling layouts with one another, it is useful to resolve the greatest discrepancy in the understanding of the space-filling property, which is the one between implicit and explicit layouts: most explicit space-filling layouts relying on successive refinement of node positions can equally be understood as implicit and thus space-filling in the usual sense. The same holds true the other way around. Examples for this duality are given in Figure 5.10, and this principle is also actively used by techniques like the EncCon Enclosure+Connection layout in 2D [NH05] and in 3D [HNLH07]. Hence, this chapter considers techniques to be space-filling, even if they do not adhere to the earlier definition, but if their implicit equivalent does.

The evaluation and comparison of the point-based layout to other space-filling layouts is done in three steps:

1. **proving the space-filling property** by showing the equivalence to the other, established space-filling methods,

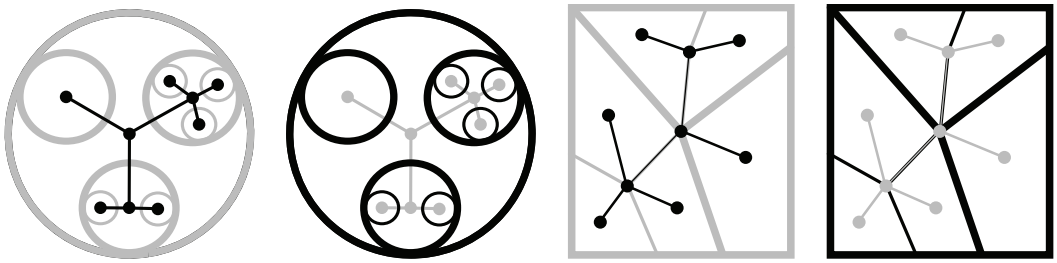


Figure 5.10: (a) The Balloon Drawings [LY07] or Bubble Tree Drawings [GADM04] are explicit techniques by design. (b) Yet, if their bounding circles are used, it becomes a circular Treemap technique. (c) The Space-Optimized Tree Visualization [NH02, NH03] has been published as an explicit technique as well. (d) The derived version uses the underlying polygonal space-division to yield an equivalent implicit layout.

2. **detailing the constraints** under which this and the other layouts become space-filling and using these constraints to classify them,

3. **comparing the space usage and readability** of this and the other layouts – quantitatively through numerical measures and qualitatively through a preliminary user study.

Proving the Space-Filling Property

To prove the space-filling property of a node placement by successive refinement of grid positions as utilized by the point-based layout algorithm and illustrated in Figure 5.2, four established space-filling approaches are given and it is shown for each case, how it can be applied to achieve the very same node positioning as the point-based layout method does. In the course of this discussion, which gives already a very strong indication for a space-filling layout, the space-filling property is also proven by computing the fractal dimension of the point-based layout.

The embedding approach. Since the point-based layout uses a fixed template of node positions and assigns them to the nodes of the tree, it is by design an instance of the embedding approach. This approach scatters possible node positions evenly across the available space, effectively creating a fixed template, and then computes a mapping of the hierarchy to be displayed onto these positions – e.g., with an algorithm like [BMS97]. Since the positioning of the nodes is independent of the actual hierarchy, it can easily be done once and for all. After that, the hierarchy to be visualized just needs to be embedded in this precomputed layout template. The evenly distributed node positions are guaranteed by the layout, as it places nodes always right between already existing nodes as seen in Figure 5.2. This observation is an important first step, as it assures an even space utilization.

The implicit approach. In the fixed template of the point-based layout, the distance between a node on level l and its children on level $l-1$ is at most $5^{-l/2} \times \textit{initial grid size}$

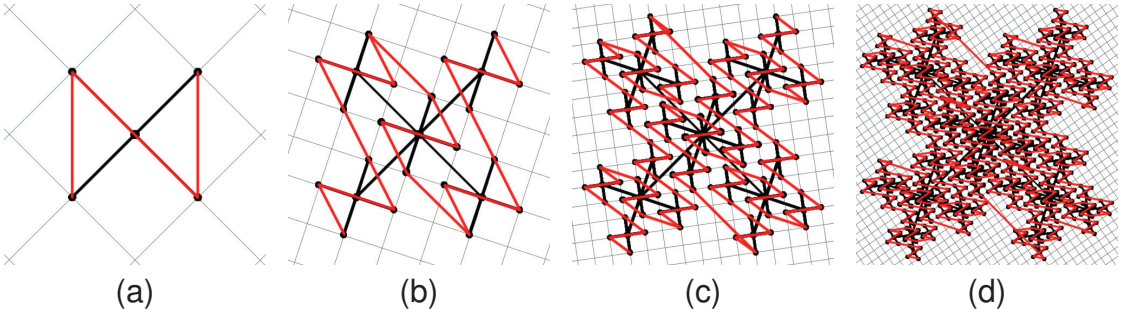


Figure 5.11: A slight modification of the Z-curve that yields the layout in Figure 5.2.

– hence, the outwards growth of the occupied area for each subtree soon falls below pixel size and thus stops. This results in boundaries emerging implicitly as the result of the reduced growth with each level. These boundaries are exactly the outlines of subtrees as described in Chapter 5.1.1. Thus, the emerging areas form an implicit, nested Treemap-style pattern: on each level, the parent area is divided in five subareas of the same shape – the outer four are used for the first four children of the parent node, the center area is recursively subdivided further using the same mechanism. If the layout would use the areas of the subtrees instead of point-primitives in their centers to represent nodes, the layout in its implicit form would be space-filling. Yet, the point-based layout itself does not define or use areas or shapes.

The space-filling curve approach. An approach that is sometimes used to generate space-filling layouts that are not implicit, is to place the nodes along a self-similar, space-filling curve [San07, MM08]. Such a curve can be interpreted as a fixed layout template and the linear node placement along the curve as an embedding by enumerating the nodes of the tree. The most common enumeration strategy is in-order DFS which places the root of a (sub-)tree as the median in the center of the linearizations of its children to preserve locality. An example of a space-filling curve that spans the point-based layout and would allow the hierarchy placement by enumeration of its nodes is given in Figure 5.11. The fact that such a curve can be established for the point-based layout is strong evidence for it to be indeed space-filling.

The fractal approach. The final proof for the space-filling property can be given by computing the fractal dimension of the layout. Fractals are often used for degree-constrained tree representations. Through constant refinement with each level, it generates eventually a closed shape, thus being space-filling – depending on the characteristics of the tree [KY93]. In the point-based layout, the self-similarity is apparent. It can even be generated using a bracketed Lindenmayer rewriting system with the configuration given in the box on the side. This configuration generates exactly the

variables: P – draw point
l – forward $\sqrt{5}$ steps
L – forward 5 steps
constants: $\oplus$ – turn 90°
$\otimes$ – turn 27°
start: P
rules: $P \rightarrow [\otimes P[lP] \oplus [lP] \oplus [lP] \oplus [lP]]$
$l \rightarrow L$
$L \rightarrow llll$

layout scheme, as it is shown in Figure 5.2. Because of this high regularity, it is easy to compute the Hausdorff-dimension D of the layout, which proves that the point-based layout fills indeed the 2-dimensional space:

$$D = \frac{\log(\textit{parts per subdivision})}{\log(1/\textit{scaling factor})} = \frac{\log(5)}{\log(\sqrt{5})} = 2$$

Detailing the Constraints of Space-Filling Approaches

While the point-based placement strategy may be space-filling in its most general form, there are often side issues involved, for example, the size and shape of the available drawing area and characteristics of the tree to display. Different layouts cope differently with these issues and some techniques remain space-filling under all circumstances, while others require certain constraints to be met:

I. **Specific aspect ratio.** Some techniques require a certain aspect ratio to be truly space-filling – e.g., an aspect ratio of 1 for PieTrees [DBW00, ODB06].

II. **Certain shape.** Other techniques may cope well with different aspect ratios, but can only fill the available screen region if it is of a certain shape – e.g., a rectangular shape for Treemaps.

III. **Unlimited screen space.** As a number of techniques grow outwards from the root node, no upper bound for their final size can be fixed in beforehand – e.g., for Sunburst [SCGM00].

IV. **Particular data characteristics.** Many techniques fill out the entire available screen space only if the given tree fulfills certain properties – e.g., a uniform height for Icicle Plots [KL83].

These four constraints can be used to classify existing space-filling techniques by the subset thereof they need to be fulfilled to remain space-filling. Using layout limitations like the ones listed above distinguishes this classification from previous characterizations like Wattenberg's layout constraints for implicit techniques [Wat05], which rather add additional constraints on top of the space-filling property, e.g., being order preserving or handling structural changes smoothly, to create a "perfect" space-filling layout. With these limitations in mind, a visualization designer who wants to use a specific space-filling technique can simply look up which constraints the display space and the input tree must fulfill to create a truly space-filling representation.

So in theory, a space-filling visualization technique might be constrained by any of the 16 possible combinations of the four constraints mentioned above. Yet, it is an interesting observation that when trying to match up techniques from the literature with the constraints above, only four of these combinations appear in practice. These four combinations are:

1. **No constraints at all.** Only very few techniques are able to completely fill an arbitrarily sized screen space of any desired aspect ratio and shape for any imaginable tree. Examples for these cases are the Voronoi Treemaps [BD05] and the Space-Optimized Tree Visualization [NH02, NH03] shown in Figure 5.10c.

2. **Constrained by the shape.** These techniques can handle it all: arbitrary trees, different aspect ratios and an upper bound on the available screen space. Just the shape of it has to be right, which usually means "rectangular". Since rectangular screen regions are the most common in today's GUIs, this constraint is not a limiting factor in practice. Besides the standard Treemap, other examples for this category include Radial Edgeless Trees [HZH07, HZ07, HZGZ09] and the Draw Tree Algorithm [GR03].

3. **Constrained by aspect ratio and shape.** In this case, not only the shape of the screen space is restricted, but also its aspect ratio. This group of techniques consists mainly of radial approaches like the circular Treemap [WWDW06], RINGS [TM02], the Balloon Drawings from Figure 5.10, and the aforementioned PieTrees. Since they all base on a circular shape, an aspect ratio of 1 is a necessity. Otherwise the techniques would degenerate into elliptic shapes, which are often considered to be techniques on their own right – e.g., ellimaps [ONF07, ONG+07].

4. **Constrained in every aspect.** This category contains all techniques that pose constraints on every aspect (shape, aspect ratio, screen space, and tree characteristics) to become truly space-filling. These are basically all the techniques which grow outwards, like Sunburst or Icicle Plots. As their shape, aspect ratio and required screen space depends on the tree itself, it is a matter of the data whether these techniques really fill out the space. In the cases of Sunburst and Icicle plots, the tree must have a uniform depth to gain a complete coverage of a circular resp. rectangular space. Additionally, for Icicle Plots, the ratio of the tree's width and height determines the aspect ratio of its icicle representation.

Interestingly, the point-based layout falls into neither of these four categories. Instead, so far it seems to be the only member of the following fifth category:

5. **Constrained by shape, aspect ratio, and tree characteristics.** This class of techniques requires a certain shape of a certain aspect ratio, which is only filled up to the last pixel, if the tree's width and/or height adhere to certain principles – e.g., being a multiple of 4. Yet, it is still possible to set an upper bound on the size, which will not be exceeded by the visual representation.

These five categories can be schematically depicted as shown in Figure 5.12 to give an impression of the landscape of currently existing space-filling techniques and the gaps between them. For example, Figure 5.12 shows that the constraint I (aspect ratio) seems to be a stricter version of constraint II (shape): while there exist many techniques that require a certain shape, but can handle arbitrary aspect ratios, there is no known technique that does the opposite – is able to fill an area of arbitrary shape but needs a fixed aspect ratio. So, a fixed shape is a necessary condition for a fixed aspect ratio – at least until a technique is developed to fill this gap.

Comparing the Point-based Layout with Respect to Space Usage and Readability

The point-based layout tries to push screen utilization as far as possible while at the same time trying to still reflect the inherent properties of the hierarchy. To achieve the latter, it

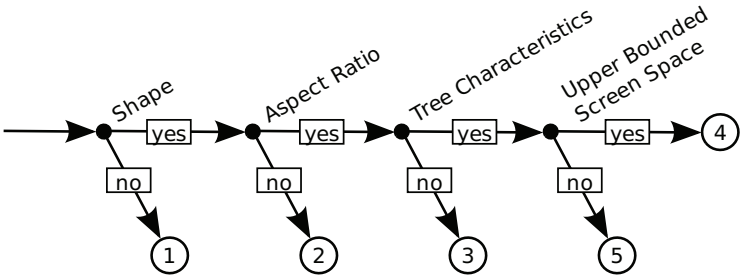


Figure 5.12: The 5 categories of space-filling techniques and their constraints.

refrains from evening out the characteristic features as some other space-filling layouts do. Yet, leaving pixels empty aggravates overplotting and thus cluttering of the representation. Hence, readability depends on the right balance between a good space utilization and a still meaningful layout that conveys the tree properties.

The following comparison of the point-based layout is done against two concrete existing node-link layouts for large hierarchies – the Space-Optimized Tree layout and the RINGS layout. Like the point-based layout, both layouts show traits of the different space-filling approaches, i.e., of the implicit approach as sibling subtrees are placed in non-overlapping areas that are nested within the drawing area of the parent. Furthermore, these layouts fall into different categories of the classification – both of which being less constrained than the point-based layout.

The following numerical evaluation will rank the three techniques with regard to their respective **screen utilization and overplotting**. For the following discussion, two quantifying measures are used: the Ink-Paper-Ratio as well as *overplotted%* from [ED06]:

$$\text{overplotted\%} = 100 \times \frac{|\text{overplotted pixels}|}{|\text{used pixels}|}$$

For the comparison, the node size is fixed to 1 pixel and a quadratic drawing area of 600×600 is used. The comparison is done for an artificial test set of hierarchies and for the DMOZ hierarchy. As an optimal space-filling layout could place at most 360,000 nodes on a 600×600 screen-space, three different full trees with nearly as many nodes were constructed. These trees are fully balanced, so that each non-leaf node has the same number of children. The chosen width-depth-combinations are listed in Table 5.1. The computed measures for the different layouts and trees are shown in Figure 5.13. The DMOZ data set runs somewhat out of the competition, as its size of 765,328 nodes

test case	depth	width	# of nodes	# of leaves thereof
A	6	8	299,593	262,144
B	7	6	335,923	279,936
C	9	4	349,525	262,144

Table 5.1: Sizes of the three evaluated trees.

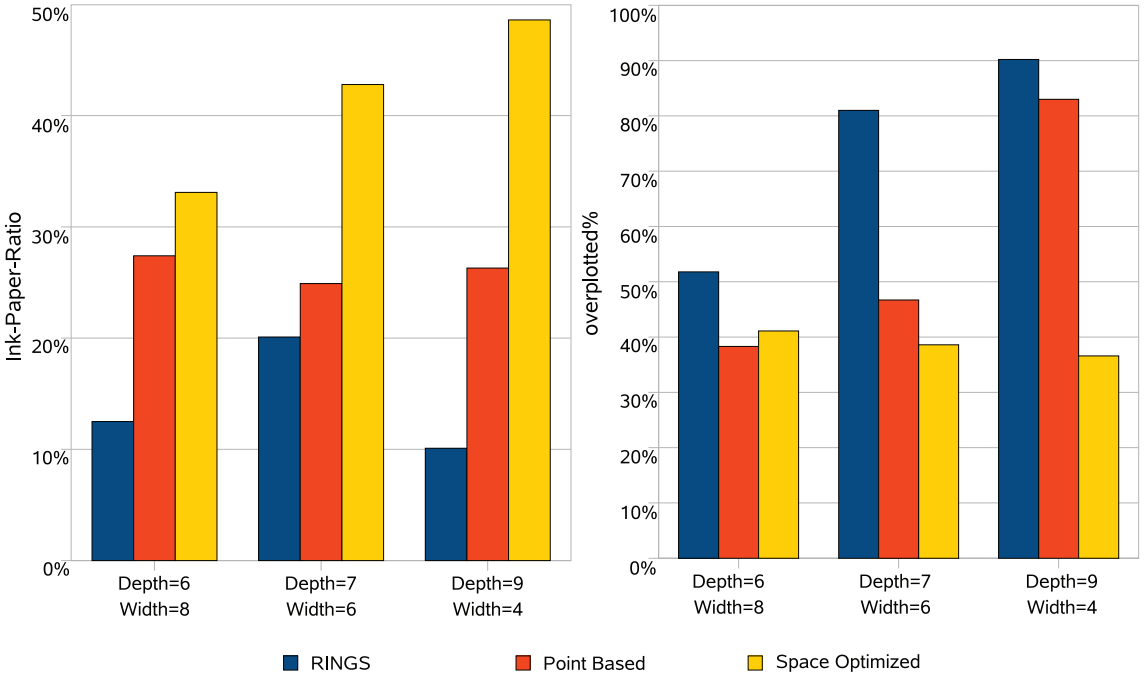


Figure 5.13: The Ink-Paper-Ratios and overplotted%-values for the three trees from Table 5.1.

exceeds by far the number of available pixels. Hence it violates condition (a) of the space-filling definition which explicitly ruled out a too small drawing area. Yet, as the conditions in real world scenarios are usually far from perfect, the numerical study of this case seems nevertheless worthwhile.

The Ink-Paper-Ratio shows the utilization of the available screen space. A higher value means a better usage of the space and is thus preferable. First of all, it is noteworthy that all three techniques behave differently with respect to increasing depth and decreasing width. While the Ink-Paper-Ratio is steadily rising for the Space-Optimized Tree layouts, it stays about the same for the point-based technique. This seems right, as the point-based layout is the one that is the most fixed, whereas the Space-Optimized Layout has complete freedom to place its nodes and even RINGS is flexible in the regard of how many children to place at each level. As for RINGS, in terms of the Ink-Paper-Ratio, it would behave more similarly to the Point-based layout, if it were not for a special case in the RINGS layout for trees with exactly 6 children for which the layout allows the sixth subtree to occupy the entire middle of the layout, which leads to the better ratio for this case. Furthermore, the left diagram in Figure 5.13 shows that the maximal Ink-Paper-Ratio is still below 50% for all of the test cases, which is already quite good for an explicit layout. It is not surprising that the Space-Optimized Tree layout achieved the best utilization for all of the trees as it is the one with the most adaptable of the three layouts examined. Yet, its adaptability, which tries to distribute the nodes evenly and thus to maximize the screen usage, leads to the effect that the overall characteristics of the tree are evened out, which makes it hardly possible to relate the density of one

subtree to another. This is where techniques like the Point-based layout or RINGS have their strengths. Here, tree characteristics are preserved through the layout process at the expense of a lower screen utilization, resulting in blank spaces that are not occupied. That the Ink-Paper-Ratio values of RINGS are below the ones of the Point-based layout is mostly due to its circular approach, which leaves a lot of whitespace by design. This is not necessarily a drawback, as it allows to clearly distinguish the individual subtrees from each other, whereas the Point-based and the Space-Optimized approach achieve a tighter packing and thus a better Ink-Paper-Ratio, but at the expense of the subtrees' distinguishability. As for the DMOZ data set, the Ink-Paper-Ratio behaves similarly as for the test cases. Only RINGS shows a comparatively low value here, as its utilization of the screen is upper-bounded by its circular layout: once the circular areas are more or less filled, no more pixels can be used for the layout. This is different, e.g., for the Point-based layout which places new pixels always in between existing ones, potentially using left over space until the procedure falls below sub-pixel resolution.

The overplotted% values give the relative amount of overplotted pixels, which is in this case equivalent to visual clutter. Hence, a lower value means less clutter and is therefore desirable. It can be observed in the diagrams that the overplotted% values of the Point-based layout and the RINGS layout are both rising with increasing depth and decreasing width. Both show also a distinct jump from about 50% to about 80%. Only this jump occurs earlier for RINGS than for the Point-based approach. Interestingly, the overplotted% values for the Space-Optimized Tree layout are even slightly falling with increasing depth and decreasing width. This is probably due to the fact that this technique is very much influenced by the width of the tree, because it partitions the available space according to the number of children. When the number of children is large, it produces many narrow partitions, which make the layout even harder at the next level. Here again, like in the Ink-Paper-Ratio diagram, the Point-based technique is sandwiched in between the other two techniques. The only exception is test case A, where the Point-based layout performed slightly better than the Space-Optimized layout. This overall distribution is related to the Ink-Paper-Ratio, since the use of more pixels has an effect on the degree of overplotting. For the DMOZ dataset, the comparison of the actual layouts hints at some interesting properties. For example, while RINGS has its advantage definitely on the aesthetic side, clutter tends to occur frequently already at higher levels of the hierarchy. This can be observed from the uniformly large font in the GraphSplat in Figure 5.14(b), as the hierarchy level is mapped onto the font size. The varying font sizes in the GraphSplats in Figure 5.14(a) and Figure 5.14(c) show a much more diverse occurrence of cluttering artifacts for the Space-Optimized and the Point-based layout. Yet, while the Space-Optimized layout tries to minimize clutter by prioritizing space utilization above everything else, its algorithm actually produces more clutter than even RINGS. This can be seen in Figure 5.14(a) and is due to the already mentioned very narrow areas that are allocated in case of a wide subtree. This also produces the alignment of the dense regions that can be observed in the GraphSplat, as the midpoints of these narrow areas, which are used as roots for the individual subtrees, are all lying side by side. Of course, the point-based layout cannot prevent clutter from occurring either. But its algorithm distributes it nicely and thus minimizes it, as it is less likely to accumulate in one region.

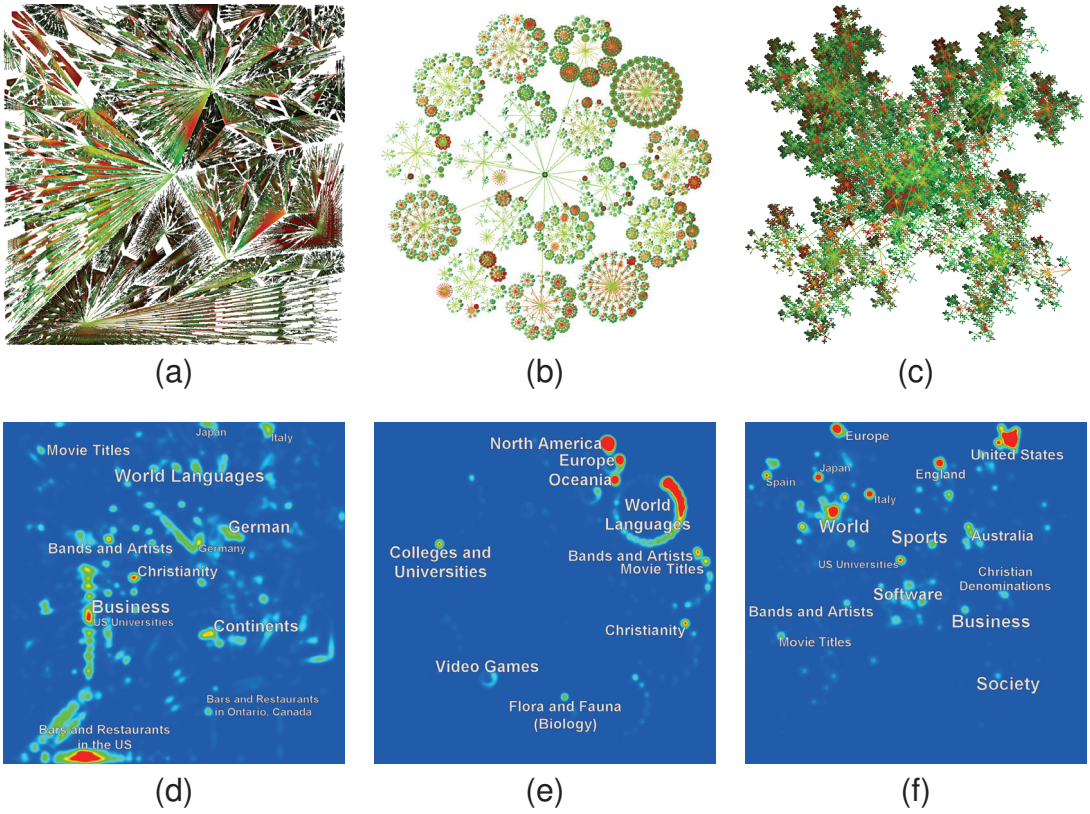


Figure 5.14: (a)-(c) The Space-Optimized Layout, the RINGS Layout, and the Point-based Layout of the DMOZ hierarchy and their respective GraphSplats (d)-(f).

Aside from numerical measures, it is of course also important to evaluate the **readability** of the three layouts by user testing. A preliminary user study was conducted with eight participants (6 non-experts, 2 visualization experts). They were presented with the DMOZ data set in all of the three layouts, each of them with the number of websites being color coded on the nodes. The user study started with a brief introduction into how the visualizations work and what the DMOZ hierarchy represents. Afterwards, the participants were asked to complete the following three tasks for each layout regarding the subtrees on the first hierarchy level:

1. identify the most unbalanced subtree
2. identify the largest subtree (in terms of number of nodes)
3. identify the subtree with the most occurrences of high attribute values

It should be noted that these tasks are far from simple, as they implicitly formulate a comparison between all subtrees of the first level and then ask to identify exactly one subtree which exhibits a required characteristic most. Noteworthy observations from this preliminary user study are listed in the following. They give some insight into which layout is best suited for which tasks and how perception is hampered in some cases:

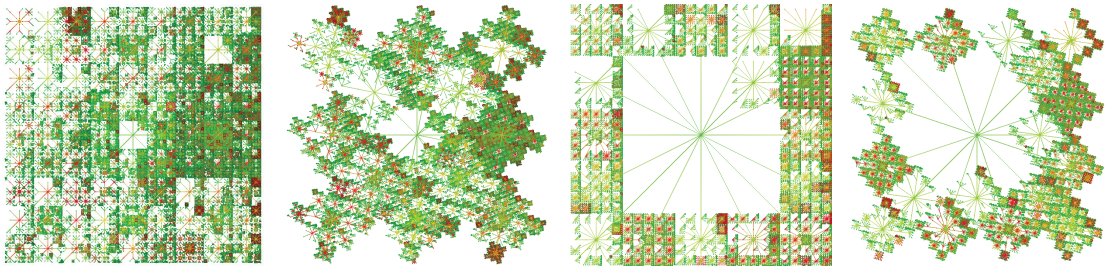
- The point-based layout performed much better than the Space-Optimized Tree layout at task 1. This is only natural as the rather even distribution of nodes hinders the perception of imbalances in the Space-Optimized layout, whereas the Point-based layout shows them.
- The point-based layout performed much worse than the Space-Optimized Tree layout at task 2. It is assumed this is, because the Space-Optimized layout scales the areas of the subtrees according to the relative subtree sizes, which the Point-based layout does not.
- The point-based layout performed about equally good as RINGS for tasks 1 and 2. It appears that wider subtrees can be determined and judged quite easily in both layouts, whereas the perception of depth is somewhat challenging in both, but actually harder in RINGS. This is due to the very fast decreasing drawing areas for subtrees in RINGS in comparison to the point-based layout. Yet a few levels deeper down, the Point-based layout runs into the same problem, which is another argument to also provide the ability to color code depth as described in Chapter 5.1.1.
- The point-based layout performed much better in task 3 than any of the other layouts. This has different reasons: for the Space-Optimized Tree layout it turned out to be the edges that occluded too much of the visualization. And for RINGS, the reason is again the very fast decreasing drawing areas, which prevent the attribute values of lower level categories to show up. Yet, the space-efficient distribution and the use of adaptive edges in the point-based layout seemed to give just the right balance between structural and node attribute information.

While these preliminary results are far from being statistically conclusive, they already indicate the point-based layout to be a good choice except for tasks involving subtree size. This is basically the price for the fixed, fractal layout. The numerical evaluation showed that except for special cases like the one with 6 children, the layout is placed well in between the Space-Optimized Tree layout (which puts a little more emphasis on space utilization) and RINGS (which puts more emphasis on discernability of the tree structure by leaving more whitespace). The feedback from the preliminary user study shows, that this may be just the right balance between both worlds.

To ensure a fair comparison, all three layouts have been used in their basic forms without further adaptations or refinements, which would have certainly been possible. Enhancements of the point-based layout that are specifically adapted to a given screen space would almost certainly have lead to a better performance of the layout especially in terms of screen utilization. Multiple such parametrizations and modifications can be applied to further adapt and enhance the layout. Some of these parametrizations are detailed in the following part.

5.1.3 Modifications and Enhancements

This third part describes modifications of the basic layout to better adapt to trees of different average width and to different aspect ratios of the available drawing area. Additionally,



(a) $\sqrt{9}$-Layout (b) $\sqrt{13}$-Layout (c) $\sqrt{25}$-Layout without and with rotation

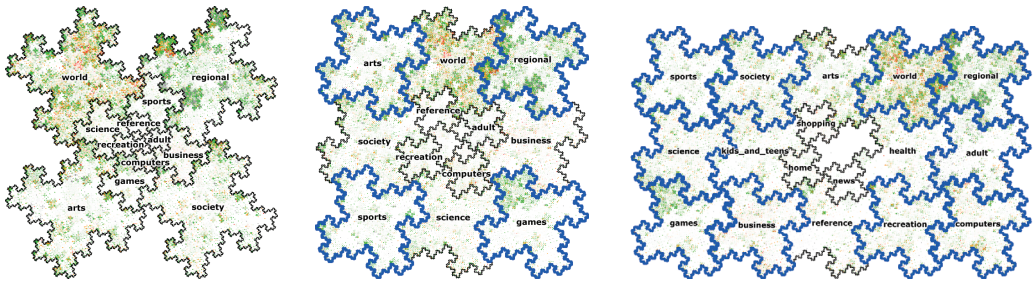
Figure 5.15: Four alternative layout methods applied to the DMOZ data set. As a side note: variant (c) is actually the Sierpinski-carpet, which has only recently been recognized as a space-filling tree visualization [MR10].

a layout enhancement is presented that allows to fit point-based tree representation into the irregular shape of regions on a map. This is achieved through taking advantage of the flexibility in node placement gained by using point primitives. While the placement of the nodes has to adhere to the predefined layout template, the template itself can be parametrized and thus adapted by a number of options.

Modifications of the Basic Layout

It is a known property of the fixed positioning of the used $\sqrt{5}$-layout that subtrees of a node are drawn at different sizes. This may become disadvantageous if the tree is very wide on average. If the average branching factor of the tree is ≈ 4, this is not a problem for the $\sqrt{5}$-layout. But if the tree is mostly much wider, many subtrees are crammed into the smaller regions around the root node, resulting in a visualization that is inappropriate for such a wide tree. One possible solution to this problem would be a reparametrization of the basic layout by using different sampling factors than $\sqrt{5}$. Because they tile the screen space into more subregions, they produce better results for wide trees. Possible variants of this are shown in Figure 5.15 using the DMOZ data set as an example. As it is the case for the $\sqrt{5}$-layout, it can be seen that a $\sqrt{x}$-layout leaves $x-1$ regions for the first $x-1$ subtrees, and keeps the one remaining region in the center for further subdivision. Figure 5.15 illustrates nicely that the DMOZ data set is not wide enough on average to make good use of the $\sqrt{25}$-layouts, as they leave a lot of empty space. And it can indeed be shown just by counting the number of used pixels, that the $\sqrt{9}$- and the $\sqrt{13}$-layout utilize about twice as many pixels than the $\sqrt{25}$-layout does for the DMOZ data set.

Another possibility to adapt the placement is to alter the first layout level by adding more regions through different tessellation mechanisms. The following levels are then laid out as usual inside these regions. This is especially effective to increase screen utilization for rectangular and square shaped drawing areas. As an example, for a square drawing area this approach already boosts the screen utilization from 60% to about 80%: four additional children of the root are laid out in the first step, the whole layout is scaled down by about 85%, and rotated 45°. Due to the interleaving structure, their subtrees integrate



(a) (b) (c)

Figure 5.16: Layout adjustment showing the DMOZ hierarchy with additional 4 (b) and 10 (c) children laid out in the first step. The children added with respect to the original layout (a) are highlighted in blue.

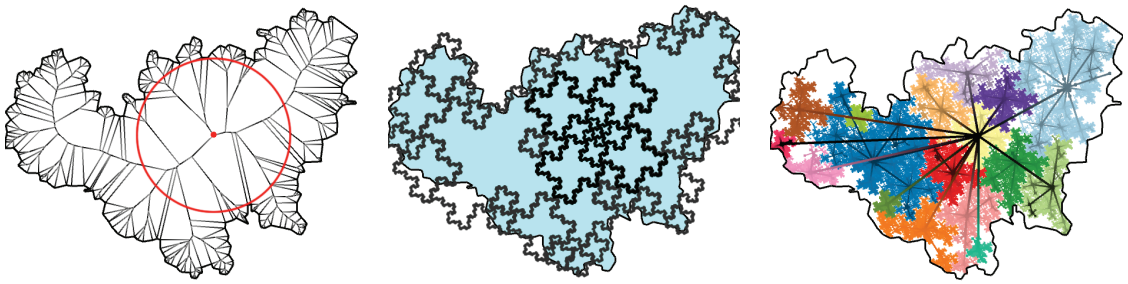
seamlessly with the overall layout. This adjustment is shown alongside the original layout in Figure 5.16(b), in which the four additional children are highlighted with blue borders. In case the available screen space is rectangular and not square, it is easily possible to apply the very same idea by adding even more subtrees on the first level, as it can be seen in Figure 5.16(c). Since this adjustment is completely independent of switching to a different sampling method because of a tree's larger width, both modifications can also be used in conjunction.

Enhancement to Fit Regions on a Map

In order to adapt the layout to arbitrary regions on maps (genus-0 shapes), a simple reparametrization of the basic layout is not sufficient. While the basic idea is the same as for the adaptation to different aspect ratios – to change the layout of the first level – more computational effort is necessary for arbitrary shapes. The proposed 3-step procedure for this is illustrated in Figure 5.17 and detailed below:

1. Computing a Center. The search for a central position for the root node is not a trivial problem as an irregular shape does not possess an obvious and easy to find center as it is the case for rectangular shapes. It should reside inside the given shape and have a maximum distance to its boundary. The straight-forward approach of computing the barycenter is problematic for concave regions, as it may be located outside of the region itself. A better choice is the center of the largest inscribed circle. This point can be computed by constructing the skeleton or medial axis of the shape. This reduces the search space for the center of the largest inscribed circle to all points that lie on the skeleton. A skeleton point fulfilling this condition can be found efficiently by computing the skeleton's importance and choosing the skeleton point with maximum importance, as it is done, e.g., in the algorithm presented in [TvW02]. This point makes a good root node position for genus-0 shapes, as it fulfills the above mentioned constraints.

2. Computing a Tessellation. Once the position for the root node is determined, the area around it can be tessellated with subregions that will later be distributed among the nodes of the first level of the hierarchy – very much in the same spirit as it was done above for rectangular regions. The first four areas are trivially placed around the root node, just



(a) Computing a center (b) Computing a tessellation (c) Mapping the tree

Figure 5.17: The 3 steps of generating a point-based layout for irregular shapes: finding a position for the root node, computing an approximate tessellation, and finally mapping the tree onto the tessellation. The colors in the resulting layout indicate different subtrees, making them discernable from each other.

so they fit inside the inscribed circle. The next areas are then placed tightly on the side of the first areas. This process starts in a greedy manner with the largest possible area and proceed recursively to fill in gaps and smaller areas step by step. As it can be seen in Figure 5.17b, the areas do neither fill every tiny bit of the shape, nor do they strictly stay inside it. That is why this tessellation is called only an approximate one. This relaxation is made possible by an adaptive mapping in the next step, which locally corrects the case where areas are lying partially outside of the shape. As long as the part lying outside is not too big, its benefit is that it generates potentially larger areas. As a rule of thumb, a portion of at least 60% of the area to lie inside of the shape has proven sufficient. If needed, the larger areas can afterwards still be divided in smaller ones, in case there are a lot of nodes on the first level of the tree.

3. Mapping the Tree. In this last step, the children of the root are assigned to the regions computed by the tessellation. Hereby, it is important to assign only to regions whose center can be connected to the root's position by a straight line that lies entirely within the overall shape and does not cross any neighboring shapes. This can be noticed for the top-left region in Figure 5.17b, which is not allocated in Figure 5.17c. If there are more children than regions, larger regions can be split further into subregions to accommodate all. If there are fewer children than regions, multiple smaller, adjacent regions can be merged into a larger one, gaining more space to layout lower level children. Then, the children of the root can be mapped onto the regions in the usual manner, placing the largest subtrees on the larger areas and the smaller ones on the smaller areas. Within the areas themselves, the basic layout algorithm Algorithm 1 is used. In the case that an area overlaps the shape, for each newly computed node position P' in Line 11 of Algorithm 1 it is tested whether it lies inside or outside of the shape. If P' lies outside, this position is skipped and the next available position is computed and tested until a position inside the shape is found. This effectively corrects the case of areas lying partially outside of the shape, which was introduced by the approximate tessellation in step 2.

The specific scenario of the shapes being rather static regions on a map allows to

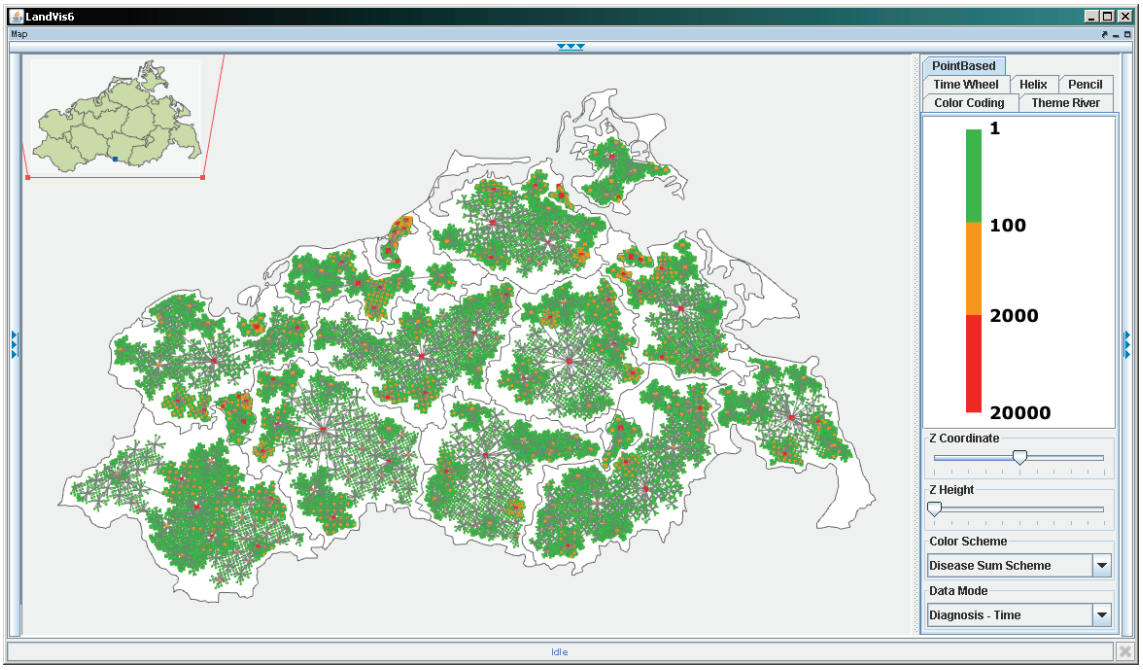


Figure 5.18: The LandVis application showing the point-based layout of the ICD disease categories in each administrative district of Mecklenburg-Vorpommern. Disease occurrences are color coded from green (less than 100 occurrences) to red (more than 2000 occurrences).

precompute step 1 and 2 as they are independent of the actual tree. Yet, this scenario brings with it another challenge: with the layout filling the regions up to their boundaries, neighboring regions often appear as if merged and are hard to discern from one another. In this case, the skeleton can be used to generate a slightly smaller shape and to layout the tree inside this shape instead – effectively introducing a narrow border for each region.

This layout adaptation for irregular drawing areas is integrated in the **LandVis** system, which is a framework for the visualization of human health data [TSWS08]. The health data contains information about how many occurrences of diseases have been registered in a certain area on a specific day. This application context contains inherent hierarchical structures to be visualized over the map: the ICD code hierarchy of diseases and the hierarchy of the time domain (years, months, weeks, days). The point-based layout is applied for both hierarchies contained in this data set by combining them to show the presence or absence of individual diseases at a given temporal resolution (usually months). To that end, the years and months are attached as subtrees to the leaves of the ICD code hierarchy and their nodes color-coded with the numbers of reported occurrences for each disease in each month – in each region. Figure 5.18 shows the adaptive point-based layout in the administrative districts of the German state Mecklenburg-Vorpommern. The tree inside each region shows the mentioned mixed ICD/time hierarchy with a non-linear, discrete color scale. Red spots in the tree indicate a high number of occurrences and may thus trigger an in-depth analysis, e.g. by brushing the attribute values and exploring them in a different

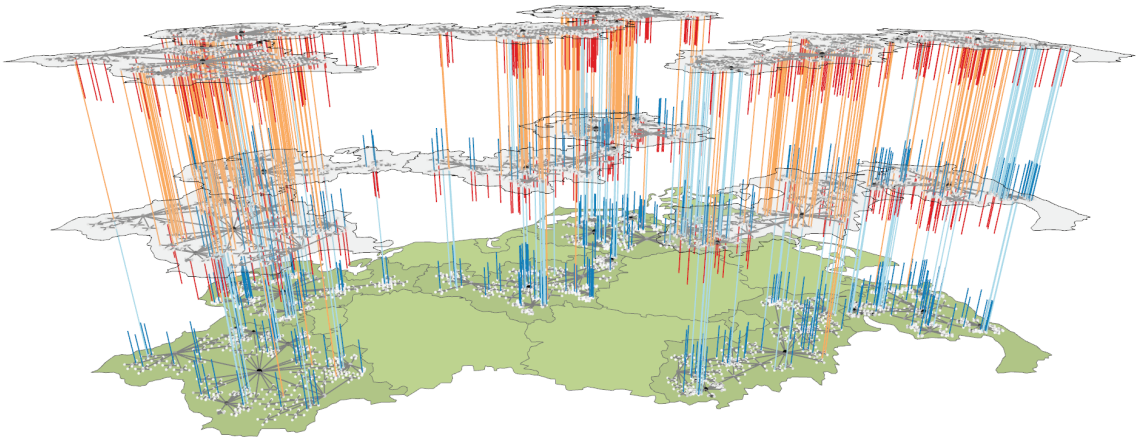


Figure 5.19: Selected regions of Mecklenburg-Vorpommern and changes of the occurrence of diseases over the years 1998 (bottom) through 1999 (middle) to 2000 (top).

view. Additionally, time can also be spread out across the 3rd dimension to better support tracing the occurrences of certain diseases over time[4] as shown in Figure 5.19. The added visual cues are straight lines that, if they connect two time points (orange lines), indicate the relocation of a subtree. If they are just stubs, they indicate an added node (red line stubs) with respect to the previous time point, or a deleted node (blue line stubs) with respect to the following time point.

5.1.4 Additional Remarks

By making a number of unconventional design choices (e.g., for a node-link layout that does not necessarily show links) and not attempting to fulfill every aesthetic layout constraint (e.g., no equal space for siblings), the point-based tree layout is able to achieve a unique combination of advantages. At the core stands the use of the smallest graphics primitive known, the point primitive, with a provably space-filling, hierarchical placement strategy. These design decisions are geared to maximize the representation's scalability, enabling it to display trees with up to a million nodes. Together with the reasonable runtime complexity of the placement, this makes for a highly **efficient representation**.

This basic setup with its fixed hierarchical placement results in a deterministic layout, that prevents the tree from evening out over the available drawing space. This is a highly desirable property for an overview visualization, as it yields a characteristic tree representation that shows main features of the overall structure or of the node attribute distribution. In combination with a tailored coloring technique and a number of appropriate interaction techniques it thus provides an **expressive overview** of large trees.

Especially for interactive data display, the fixed layout can play to its strength and allow

[4] supervised diploma thesis "Visualisierung von hierarchischen Strukturen auf einer Karte mit dem Point-Based Layout" by Steffen Hadlak 2009, to appear in part as "Visualization of Hierarchies in Space and Time" at the GeoVA(t) Workshop on Geospatial Visual Analytics: Focus on Time 2010, with an extended version invited for the International Journal of Geographical Information Science

for a very **effective exploration**. It provides orientation while zooming or rotating and even across multiple time-points in time-varying hierarchies, as shown in Figure 5.19. The evaluation made clear that it is a thin line to walk between efficiency and effectiveness. The transition between a well discernable, but less space-efficient layout (e.g., RINGS) and a less discernable, but highly space-efficient layout (e.g., Space-Optimized Tree layout) is quite fluent and the optimal balance in between is hard to pinpoint. Yet with the point-based layout, it seems that a good combination of both worlds was found.

The following chapter presents an equally unique, novel visualization technique – this time with a focus on a specific graph class: bipartite graphs with many node attributes.

5.2 A Table-based Visualization for Bipartite Graphs

As mentioned in Chapter 2.1, bipartite graph structures commonly occur in many application fields, including the graduate school "dIEM oSiRiS" in the context of which the work for thesis was conducted. In contrast to that, their visualization seems still underdeveloped – especially for larger bipartite graphs with some thousands of nodes. Displaying bipartite graphs of this size truthfully, meaning with a clear distinction between both node sets, is a challenge. On top of that, multiple node attributes and additional edge-sets, so-called *1-mode projections* also need to be integrated in the visualization.

These 1-mode projections are a peculiarity of bipartite graphs. They are computed by adding an additional edge for all paths of length 2 in the bipartite graph. These projections are not mere theoretical constructs, but actually commonly used to simplify bipartite networks and to utilize these simplified networks, the projections, for exploration. A commonplace example is the one of an author-publication network, with scholarly authors and publications as node sets and an edge connecting an author with a publication if the author (co-)authored it. A 1-mode projection of this bipartite graph onto the set of authors would add a link between two authors, if they coauthored at least one publication. The actual number of coauthored publications would be encoded as edge weights of the projected edges. Hence, for means of social network analysis and other exploration of the authors only, such a projection is very helpful. The same goes for the projection onto the publications, e.g., for purposes of document classification where it might be enough to know if and how many of the same authors have worked on a number of publications together.

To design a suitable visualization depicting all these graph-class-specific parts of the data is certainly a challenge. As a solution, this chapter utilizes a layered visualization technique for all the different data aspects to display, positioning them in parallel parts of the screen. The layered design approach has received some attention in recent years. It has not only been used for visualizing graphs, e.g., transition graphs [PvW08], but also multi-variate data in general [SGL08], and even tag clouds [CVW09]. In order to use it for displaying bipartite graphs, a tabular display mechanism is employed, that shows each node as a row in one of two tables of node sets placed side-by-side. The layout resulting from this approach has several advantages:

in terms of efficiency – as the layout relies on the fast and simple linear node placement using tables, which in turn scale up to about 100,000 nodes [EK02].

in terms of expressiveness – because the layout using two tables does not only capture the very essence of bipartite graphs, but also generates a very clear and uncluttered representation.

in terms of effectiveness – since the tabular display provides means to (re-)order the node sets, constraints the search space to either scrolling up or down, and is familiar to any user of spreadsheet software.

The fundamental, table-based layout and its basic interaction techniques are detailed in the following. Thereafter, a script-based technique for selections across both node sets is introduced. It complements the basic layout and its interactive features by adding a way to automate tedious topology-based selections of multiple nodes, as well as to store and share them across multiple software instances and data sets. In a last part, three visualization examples from the domain of life sciences are given: a biomedical ontology, a biochemical reaction network, and a time-varying simulation trace of a polymerization model.

5.2.1 The Visualization Technique

The core visualization technique consists of five parts laid out in parallel as mentioned: the two node sets, the edge set, and the two 1-mode projections. At its very basis, the visualization consists of two tables that represent the two independent node sets. Both tables have columns for the node attributes of the respective node set that they are displaying. Each row represents one node and displays its node attributes. The connecting edges between the two node sets are represented as straight lines, which run from a node's row within one table to a row of the other table. Edge weights are encoded into the width of the lines. The projections are depicted as arcs at the far side of the two tables. The overall layout of the described tables, edges, and arcs is shown in Figure 5.20. It is apparent, that such a layout can be generated quickly and adapted equally fast upon changes in the data. It runs in $\mathcal{O}(n \log n)$ if the tables need to be sorted, otherwise in $\mathcal{O}(n)$.

The main idea behind this table-based setup is to provide an attribute-centric visualization for large bipartite graphs. The tabular arrangement of the nodes gives the overall visualization an orderly and uncluttered appearance, which stays that way no matter how large the graph is. Yet, the limits of the tabular display approach become apparent when interacting with a larger table: if it grows to an order of magnitude of about 100,000 nodes, it becomes barely usable because of navigation issues [EK02]. At that point, scrolling the table becomes very time-consuming and tiring and it is in general not possible anymore to get a quick overview of the entire graph. Hence, as visual clarity is an inherent feature of this kind of display, the main aim of the visualization utilizing the table-based layout is to enhance its navigability in all aspects: for large node sets, for large edge sets, and for large attribute sets. These enhancements are detailed in the following overview of additional features which have been included to enhance the accessibility and to maintain a navigation with ease.

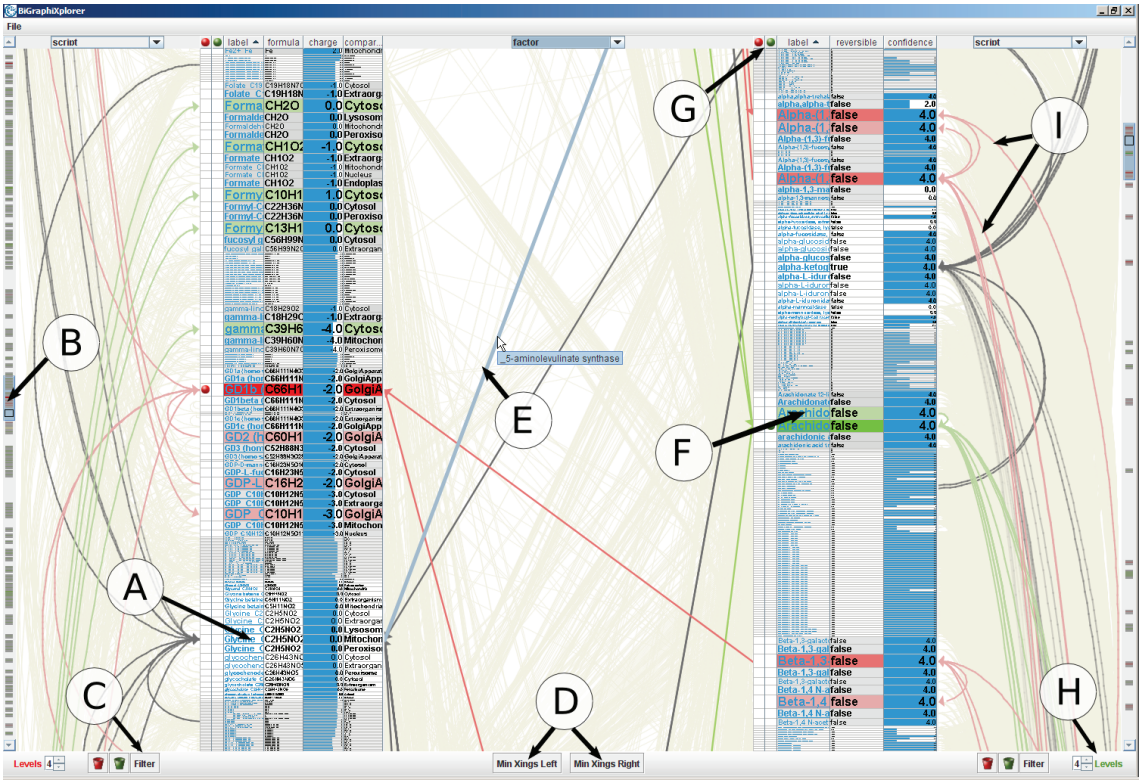


Figure 5.20: This screenshot shows the two node sets as tables, the connecting edges in between, and both 1-mode projections at the sides. The markers point to the special visualization features that were added to the basic concept. (A) - Focus+Context in table, (B) - Fisheye scrollbars with selection markers, (C) - Hide unselected rows, (D) - Minimization of edge crossings, (E) - Clickable edges, (F) - URL-references, (G) - Columns for the two different selections, (H) - Maximum level of script, (I) - Highlighting of traversed edges and 1-mode projections.

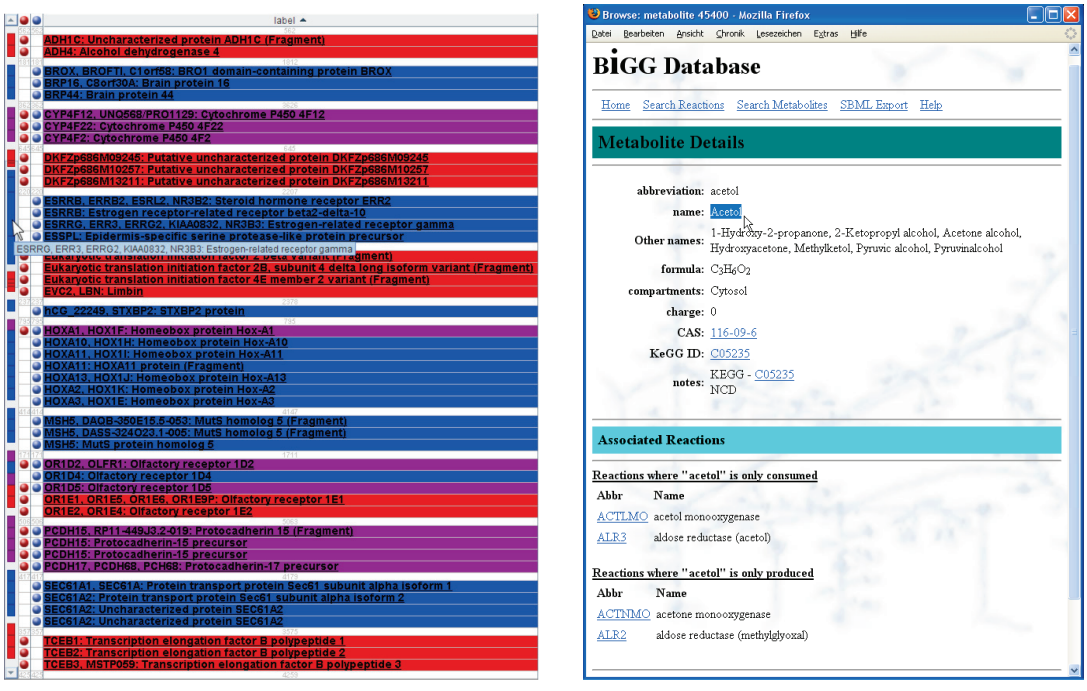
Large Node Sets

Large node sets result in very long tables, which are tedious to browse and navigate. There are two fundamental ways to cope with this – either by compacting the representation in order to reduce the table's height or by introducing interaction methods that allow to quickly navigate even large tables.

Reducing a table's height is achieved by downsizing or even hiding rows that are not currently in the focus of the exploration. Scaling down the height of rows not being focused can be done by means of focus+context techniques like the table lens [RC94] (Figure 5.20 A). It minimizes all rows that are not part of the current region of interest, which is defined by the position of the mouse cursor. The row under the cursor, as well as its neighboring rows, will be zoomed and can then be read and investigated. As for the second option of hiding rows completely, individual rows of interest can be selected and marked for later in-depth investigation. Once, all rows of interest are selected, the tables

can be adapted to show only the selected rows and filter out all others (Figure 5.20 C). In this condensed view, filtered rows will be substituted block-wise by a single row that gives information about how many rows have been hidden at that point. An example is given in Figure 5.21(a), where only the selected rows of a table are shown.

Enhancing the interaction methods to allow faster access to individual rows in large tables is another way of targeting this challenge. While this cannot be provided for all of the 10,000s of rows, at least between rows that are selected, the visualization provides navigational shortcuts by adding clickable selection markers to the scrollbars for both tables (Figure 5.20 B). Each of these scrollbars serves as an overview for one of the node sets and the added selection markers indicate focused rows even if they are off-screen. A simple mouse click on one of these selection markers and the respective table jumps instantly to the corresponding row. Because the selection markers can be placed quite densely and are hard to pinpoint for clicking, a fisheye lens is added to the scrollbar. This lens follows the mouse cursor and spreads out the focus area so that, even in crowded regions, individual selection markers can be clicked on. A tooltip displays information about the row to which a selection marker belongs. This feature is also shown in Figure 5.21(a).



(a) (b)

Figure 5.21: (a) A table in which unselected rows are collapsed into placeholder rows. The example also illustrates the possibility to adapt the color scheme to the user's personal preferences. (b) Linked HTML page - the hyperlinked name of a focused node was clicked such that a browser starts showing an HTML page with secondary attributes.

Large Edge Sets

Large edge sets result in a lot of lines running in between the two tables, which in turn produce a lot of edge crossings. This makes it hard to discern the edges and increases the cognitive load for the user, which makes it harder to follow the course of an individual edge or a path of edges. Again, as before for large node sets, solutions can be found by adapting the representation by reducing the edge crossings or enhancing the interaction capabilities of the visualization by aiding in navigating edges, even in crowded layouts.

For adaptations of the representation, one has to note that the arrangement of the edges depends on the row ordering of the nodes in both tables. Hence, in order to minimize the number of edge crossings and thus to make them easier to follow, the nodes must be reordered. This may not be possible at all times during an exploration, as a table may have been purposefully sorted by some attribute. Yet, if at least one of the tables can be reordered, a reduction of edge crossings is possible. To that end, a barycentric crossing minimization heuristic [JM97] can be triggered with just one mouse click to rearrange one of the tables (Figure 5.20 D).

The interaction on the visualization is enhanced by providing edge-based navigation: it makes it possible to move the mouse over an edge of interest that in turn gets highlighted and becomes clickable if one of its end nodes lies outside of the screen. Additional information about the off-screen endpoint is displayed in a tooltip and a simple mouse click carries the user to the off-screen node without the need to discern and follow it in between all the other edges (Figure 5.20 E). By the use of the 1-projections, it is even possible to travel directly to a node being two steps away. This speeds up the traversal of longer paths immensely, if the properties of the intermediate nodes in the other table are not of interest and do not need to be inspected along the way.

Additional Features for Large Attribute Sets

Large attribute sets with dozens of attributes per node, as they occur in real world data, result in very wide tables with equally dozens of columns. Since yet another scrolling axis by allowing horizontal scrolling would greatly reduce the ease of interaction, the visualization is limited in its width by the current screen width, whereas it can be arbitrarily extended downwards. As columns are hard to condense within the representation, as it was done for the rows, a selection of attributes has to be made on which the subsequent exploration focuses on.

In this case, the idea is to divide the attributes into primary, relevant attributes and secondary, supplementary attributes. Primary attributes are shown in table columns, whereas secondary attributes are out-sourced to HTML pages, from where they can be retrieved on demand. A mouse click on the hyperlinked name or label of a node (Figure 5.20 F) opens up an internet browser window that displays the HTML page, as seen in Figure 5.21(b). This feature is especially useful for attributes like lengthy texts, images, or other multimedia content, which is hard to fit into a table cell anyways and for which a sorting (one of the main features of the tabular display) does not make sense.

5.2.2 Topology-based Selection Mechanisms for Bipartite Networks

Especially in large bipartite networks, the ability to derive a subset of the entire data set is important for further analysis or in-depth visualization. This can be done via interactive selection – yet, especially for larger selections, this becomes a tiresome task. Hence, a somewhat more sophisticated selection scheme than subsequent mouse clicks is devised. Overall, the mechanisms for selecting subsets of data can be described along the following lines:

Addition vs. deletion: These are the two dual modes of constructing a subset. One can either start with an empty subset and add the desired data values, or with the complete data set and consecutively delete undesired data values. For interactive selection, often both modes are combined to allow for the refinement of a previously constructed subset. The toggle-functionality can be seen as the simplest example for that, but also more complex mechanisms have been shown to be useful [Wil96]. The selection scheme presented here makes use of selection by addition, which is the more intuitive selection mode and thus applicable to real-world scenarios without imposing a steep learning curve on the domain expert. This is a very important consideration, for raising the acceptance within an interdisciplinary group of researchers like the graduate school "dIEM oSiRiS" for which this visualization was specifically developed.

Automatic vs. interactive: A subset can either be constructed automatically or be specified interactively by the user. In visualization, selection is generally understood as the interactive counterpart to the more automatic, preprocessing-oriented sampling (constructed by addition) and filtering (constructed by deletion). The selection mechanism introduced here actually uses a hybrid, 2-step selection that divides the selection process in an interactive part and an automatic part. This allows to combine the best parts of both worlds: the possibility for the user to influence the selection by interactively triggering and parameterizing a complex automated selection logic, which is described in detail in the following.

Binary vs. discrete/continuous: In the binary case, a data value either belongs to a subset or not. Whereas in the discrete/continuous cases, each data value is mapped to a Degree of Interest (DoI) value [DH02], which determines "how much" a data item belongs to a subset. These kind of selection methods come naturally with focus+context techniques. Here, the focus region belongs to the selection and is assigned the highest DoI. Whatever lies outside of this focus region is mapped to a DoI below that value. When defining such a mapping, there is usually some notion of relatedness to the focus region involved. This relatedness can simply be spatial proximity in data or image space [Fur86], or more complex semantic concepts like similarity [NH06] or relevance [SCH+06]. The selection procedure presented here uses a discrete DoI, based on the user's interest in specific parts of the subset and a topology-based propagation of DoI values.

In order to realize such a propagation of DoI values, structural/spatial proximity has to be defined. For trees and hierarchies, selection mechanisms accounting for structural

relatedness have already been introduced [FWR99]. In their case, a number of different parameterizations allows to refine the scope of the selection within the tree. As structural selection is taken further to the more complex case of bipartite graphs, the number of possible configurations in the selection scope becomes even larger. This is because there is more than one way of being related in a bipartite graph: being adjacent through the edge set, being adjacent through one of the 1-mode projections, or even not being directly adjacent, but still being connected through a path of two other nodes. The structural selection method adapts the view of spatial proximity in the graph structure as shown by these examples. This means that the distance of a node to a focus node is measured within the graph structure itself and not within their screen representation. Thus, two nodes might have been laid out far apart on the screen, but if they are connected by an edge, their structural proximity is still considered high.

To make the best use of the possibly complex configurations resulting from this graph theoretic notion of proximity without reducing the ease of use of the selection, the selection process is split into an interactive and an automatic part. For this, the user selects a set of *focus nodes* in an interactive first step. These are passed to the automated part, in a second step, which successively gathers all nodes that can be reached by a predefined, set-based selection logic.

1st Step: The interactive selection. A toggling mechanism was chosen to mark focus nodes, as it can be seen in Figure 5.20. Additionally, the user can just click and drag along the selection columns to select entire regions or intervals of rows. Together with the ability to sort the table with respect to any attribute, this provides a brushing mechanism that allows to quickly select items within certain attribute ranges. To enable the comparison of results from two different selections, two columns are added in both tables (marked by (G) in Figure 5.20), where focus nodes can be picked independently. A unique color is assigned to each of the two individual selections and everything related to them (icons, menu entries, highlighted rows and edges, etc.). By default, red and green are used, as biologists are familiar with this color scheme. If it happens that a node is affected by both selections, the color yellow is used for its respective row in the table, which is inspired by traffic lights, where yellow lies between red and green. However, the colors can be adjusted according to the user's preferences.

2nd Step: The automatic selection. The result of the interactive selection in the first step is passed to the loaded selection script. Starting with a set that contains only the focus nodes, the scripted selection logic proceeds to add new node sets script line by script line in a breadth first manner by traversing along edges or projections. The maximum number of selection levels is determined by the number of lines within the selection script. It can be interactively lowered from within the visualization for both available scripts, as shown in Figure 5.20 at marker (H). The selection logic allows to define a DoI value in every script line, which is then assigned to the node set added to the selection by the script line. This way, the user's interest in the result of individual script lines can be specified more precisely compared to an automatically derived DoI value, for example by proximity to the focus nodes. The values lie between 0 and 100, where 0 means no accentuation or amplification of a row, and 100 means maximum accentuation. The value ranges have been chosen because it was found that the "percentage-thinking" is intuitively understood

throughout all application domains. In the table-based visualization, the DoI values define how the nodes of the set should appear in the tables: it is used to define the color saturation and the height of the respective row. An example of such a selection script is given in the context of the examples discussed in Chapter 5.2.3.

To make the concept more flexible, it is extended to allow the filtering of node sets by the nodes' attributes values. These can be attributes from the dataset itself or some of the commonly derived structural attributes, like indegree, outdegree or the clustering coefficient, which is used in its adapted version for bipartite graphs [RA04]. The filtering of node sets can be used to narrow down the traversed paths to only those paths of interest. The paths along which the script traverses the graph are highlighted in the visual representation of the bipartite graph (marker (I) in Figure 5.20), the edges along the paths are colored with the same saturation as the nodes they emanate from. This is very helpful for understanding the inner workings of selection scripts, as one is not only given the script result (nodes with DoI values attached), but also some information on how this result came about. It ties in very nicely with the possibility to interactively define the maximum level up to which a script is computed.

5.2.3 Applications

This part will give an overview of the usage of the visualization and selection technique within the graduate school "dIEM oSiRiS" by discussing three real world data sets. The first data set, a biomedical ontology, is foremost used to describe the visualization technique, and the second data set, a biochemical reaction network, is then used to illustrate the selection mechanism. The third data set is a simulation trace of reaction networks. It shows how the table-based visualization integrates with other visualizations in an actual exploration walk-through.

A Biomedical Ontology

Ontologies are a collection of interrelated concepts or terms that aim at providing a basis for knowledge management in a certain area. Data sets can use the provided terms, to declare their data instances of these concepts, annotating them and thus providing semantics alongside the pure numbers. These annotations form a bipartite graph, with the terms as one node set and the instances as the other. This bipartite model is well known from research on Semantic Web technologies like folksonomies or lightweight ontologies.

One of the largest biomedical ontologies freely available today is the Gene Ontology Database [ABB+00], which was used together with one of the largest annotated data set of gene products, the Homo Sapiens annotation from the Gene Ontology Annotation Database [CBDL06]. This data set consists of 26,389 terms from the ontology (OBO snapshot from 12-JUN-2008) and 35,043 gene products annotated with these terms (GOA snapshot from 07-JUN-2008). These annotations are encoded in 377,132 undirected links in between these two sets. By projecting these annotations onto the set of terms, 122,379 additional edges are produced. The projection onto the gene products ran into a combinatorial explosion and produced literally billions of additional edges. Since this number is well beyond the scope of the visualization, the huge set of projected edges was filtered

down to the 291,586 edges, with weight 16 or higher. The weight of 16 is basically an arbitrary choice, as it presents the lowest filtering threshold for this data set that yielded an edge set with less than a million edges, which is more or less the limit of the implementation. The data sets also have a variety of attributes, as short natural language definitions for the terms and bibliographical references for the gene products. To make all of this information available, the terms have been hyperlinked to the Gene Ontology Database and the gene products to the five databases they originally stem from: UniProtKB, RefSeq, ENSEMBL, H-invDB, and VEGA. So, this is a good example of how the visualization ties together individual biological databases.

When exploring this rather large data set with the table-based visualization, it soon becomes apparent that scrolling through the tables is a very time consuming and slow method to access the network. It actually takes more than 700 turns on the mouse wheel to scroll from the top of the terms table to its bottom. So, this is exactly one of the cases, where all the other means of rapid navigation come into play: clickable edges, fisheye scrollbars and folding of unselected nodes to compress the view. As shown in Figure 5.21(a), the folded view of the terms table compresses several thousand unselected rows into one-row-placeholders. Navigating in the compressed view and only switching back to the full view if a new selection is to be made, speeds up the exploration immensely.

The ontology data set exhibits an interesting way of using the projections as indicators of similarity. That is because the number of shared terms is often taken as a measure of similarity for two instances. The more instances two terms have in common, the more alike they probably are. This notion is captured exactly by the projections and their weights: the higher the weight of a projected edge, the more disjoint paths of length 2 exist between its two incident nodes. And the more disjoint paths there are, the more joint intermediary nodes must lie within the other node set. Hence, clicking and thus following the projections during the exploration makes it very easy to find similar nodes within this large data set.

A Biochemical Reaction Network

Reaction networks are usually described as hypergraphs, in which each directed hyperedge encodes a biochemical reaction connecting its reactants with its products. A common transformation of hypergraphs into the so-called König's representation [TZB96] was used to map the reaction network to a bipartite graph. Hereby, the hyperedges (reactions) are transformed into nodes themselves and new directed edges are introduced from each reactant to its reaction and from each reaction to its products. This transformation yields two node sets: the chemical substances (also called *species*) and the reactions.

As one of the largest biochemical reaction networks that has been reconstructed so far, the human metabolic network [DBJ+07] from the BiGG Database [SBRG] was used. It comprises 2,764 chemical compounds, 3,311 chemical reactions and 17,519 directed links with stoichiometric factors encoded as weights between compounds and reactions (SBML snapshot from 20-DEC-2007). Additionally, the projection on the compounds yields 121,118 edges and the projection on the reactions 295,922 edges. The data set includes an abundance of attributes, from charges for the compounds to bibliographical references for most of the reactions. As not all of them could be displayed, the names of

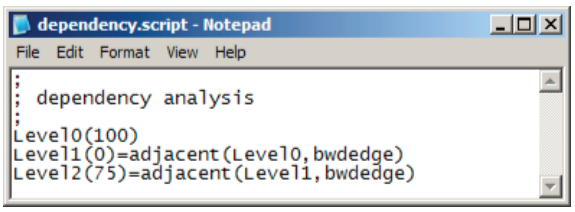


Figure 5.22: An example of a selection script.

the compounds and reactions are hyperlinked to the BiGG database with all the additional information, as seen in Figure 5.21(b).

To illustrate the script-based selection mechanism, a selection logic for a dependency analysis was derived, as this kind of analysis is often performed by biologists on reaction networks. This means, for a focused chemical compound the selection script should automatically determine on which other compounds it directly depends. As there are no edges between the nodes of one node set, direct dependence means that there exists at least one directed path of length 2 between them. Such a script must follow down all reactions that produce this very compound and select their reactants. This selection logic must be transformed into a set-based notation, which looks like the following, with $V = V_1 \cup V_2$:

$$
\begin{aligned}
&Level_0(\text{DoI} = 100\%) = \text{ focus nodes} \\
&Level_1(\text{DoI} = 0\%) = \{u \in V : \exists (u,v) \in E : v \in Level_0\} \\
&Level_2(\text{DoI} = 75\%) = \{u \in V : \exists (u,v) \in E : v \in Level_1\}
\end{aligned}
$$

Here, with each step, the logic is traversing backward edges, starting from the focused compound and traversing backwards twice: once to reach all reactions that produce the compound and another time to reach their reactants. Thereby, it is guaranteed to yield only nodes that are also chemical compounds in $Level_2$. The example also shows that the DoI is not necessarily decreasing with each level. Instead, each level can be assigned its individual DoI according to the exploration goal. In this example, only compounds but not the intermediate reactions are of interest. Hence, the intermediate $Level_1$ is assigned a DoI of 0. Given the intended logic in a set-based notation, it can easily be translated into a selection script as shown in Figure 5.22.

Once defined, this script can be loaded and used with a mouse click on whichever compound the biologist likes to run the dependency analysis on. This is the interactive part of the selection, which toggles nodes to build up a set of focus nodes to be passed to the script for dependency analysis.

It would also be possible to shorten the shown selection script by traversing the backward 1-projection on the compounds in just one step, instead of going two steps along the edges as before. This illustrates nicely how 1-mode projections indeed form shortcuts. It is, of course, possible to create much more complex selection scripts, as it was done for the example in Figure 5.20, which shows a part of the metabolic network.

Simulation Runs of a Biochemical Model

Exploring biochemical reaction networks as static "maps" can already reveal important dependencies and network properties. Yet, another important aspect is exploring their dynamics. This can be done through modeling and simulation approaches, generating sequences of bipartite graphs. In this example, a model of polymerization (monomers binding together to form polymer chains) is investigated which is formulated as communicating processes using the *Attributed π-Calculus* [JLNU08] and subsequently simulated[5]. The resulting simulation trace consists of a reaction network for each time point, each containing all possible binding reactions and release reactions – their opposites, at that point of the simulation. The transitions between them mark the occurrence of a reaction, leading from one network to another.

Visualizing this data for debugging and validation purposes is mainly aimed at simultaneously giving insight into changes of structure and attribute values over time, but also to allow to inspect cause and effect of such changes at individual time points. Only by gaining insight about the reason for an unexpected behavior that occurred, the domain experts will be able to localize and fix the bug causing it. Bringing all these requirements under the hood of one visualization is hardly possible, and tying it all into one table-based visualization would completely overburden the visualization. Yet, the table-based visualization has already shown to be good at visualizing reaction networks and aiding in their visual exploration through its numerous interaction mechanisms. As each time point constitutes a reaction network, the table-based visualization is well applicable to displaying them. For the overview of the time line of events and thus changes in structure and attribution of the reaction network expressing the current system state, a different visualization is needed. If the change in the network structure could be expressed as a value, a simple time-value-plot would be feasible, which is probably known to any domain expert and even non-experts. Especially for an overview visualization, which will be the starting point for any exploration, a well known and easy to interpret plot is an important design criteria. Merging both concepts and linking both – time-value-plot overview of the entire time line and table-based detail views of individual time points – results in a powerful, yet easy to use combination [Ung10, ch.4.2].

As for the remaining question of how to condense the structural features into numerical values to summarize the evolution of the network, multiple approaches are available – depending on which features are to be captured numerically. In the following, structural complexity measures [DES09] are used, as they quantify multiple facets of a graph's structure within one value. For its very fast computation, the example shown in this part uses the information content I_{vd} of the vertex degree distribution [BR05], which is one such possible complexity measure:

$$I_{vd}(V) = \sum_{i=1}^{V} \deg(v_i) \log_2 \deg(v_i)$$

An example of a visual exploration of a concrete simulation run of the said polymerization model is detailed below. It consists of three steps: (1) inspecting the overview and

[5] submitted as "Constructing and Visualizing Reaction Networks from Pi-Calculus Models" to Theoretical Computer Science

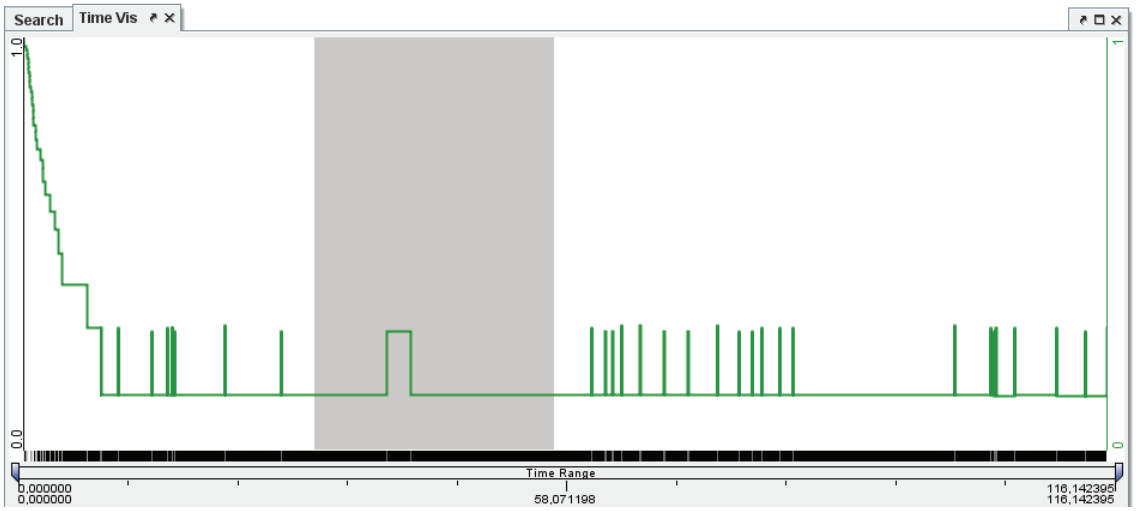


Figure 5.23: Overview showing the structural complexity of the evolving reaction network over time. The gray region is a lens region with a horizontal distortion, so that release and binding events which are otherwise condensed into spikes become visible as separate time points.

observing unexpected behavior at certain time points, (2) investigating these time points for an explanation, and (3) confirming the found explanation by comparing the reaction network of this time point with those of its preceding and following time point.

1. Inspecting the overview: Two observations can be made from the overview of the example simulation run shown in Figure 5.23: it is rapidly decreasing in the beginning and then oscillates roughly between two values. It is important to note, that the more monomers are bound, the less complex is the reaction network, because less binding reactions are possible. Hence, the plot shows a plausible system state trajectory: initially, all monomers were free, thus allowing many possible reactions among them, and therefore exhibiting a very complex reaction network. The more of them bind together, the fewer reactions are still possible and hence the lower the structural complexity of the reaction network gets. From the curve alone, one can determine that the simulation was most likely parametrized in a way that the binding probability is much higher than the probability for a monomer to be released. This would explain the monotonous decline of the network complexity, which would otherwise be highly improbable. This also explains the oscillation: when a monomer is released (spike goes up), it is almost instantly bound again (spike goes down) because of a much higher binding probability. The short time interval between release and binding is made visible by a horizontally distorted, gray region of the time-value-plot, as shown in Figure 5.23.

2. Investigating time points: The cause of the oscillation can be investigated in detail by looking at the table-based view of the reaction network for the time point of the sudden increase of complexity, as shown in Figure 5.24(a). Upon inspection of the network, the monomer just having been released is easy to identify, because it is the only present monomer (amount > 1) with its first binding site not being allocated (attribute1 = free).

One can easily see the large number of outgoing edges to possible binding reactions, which are the reason for the increased structural complexity at that point of the system trajectory in the overview.

3. Confirming by comparison: For further confirmation, this time point can be compared to the time points before and after this one. Both are shown side-by-side in Figure 5.24(b). The time point before the peak clearly shows the still bound monomer (blue) and its soon to happen release reaction (violet). As it can be seen, the products of this reaction are two monomers, one of them (the red one) the unbound monomer from Figure 5.24(a), at this time step still with amount = 0, as the monomer is not yet released. It is easy to see, that while a monomer has many binding reactions, it has only a single release reaction – hence the difference in structural complexity. The same goes for the time point after the release: the free monomer (now having amount = 0) has bound to the polymer again and now allows only one release reaction instead of multiple binding reactions as before, effectively causing the drop off in structural complexity. This confirms the assumption that alternating release and binding reactions cause the observed oscillatory pattern in the complexity plot.

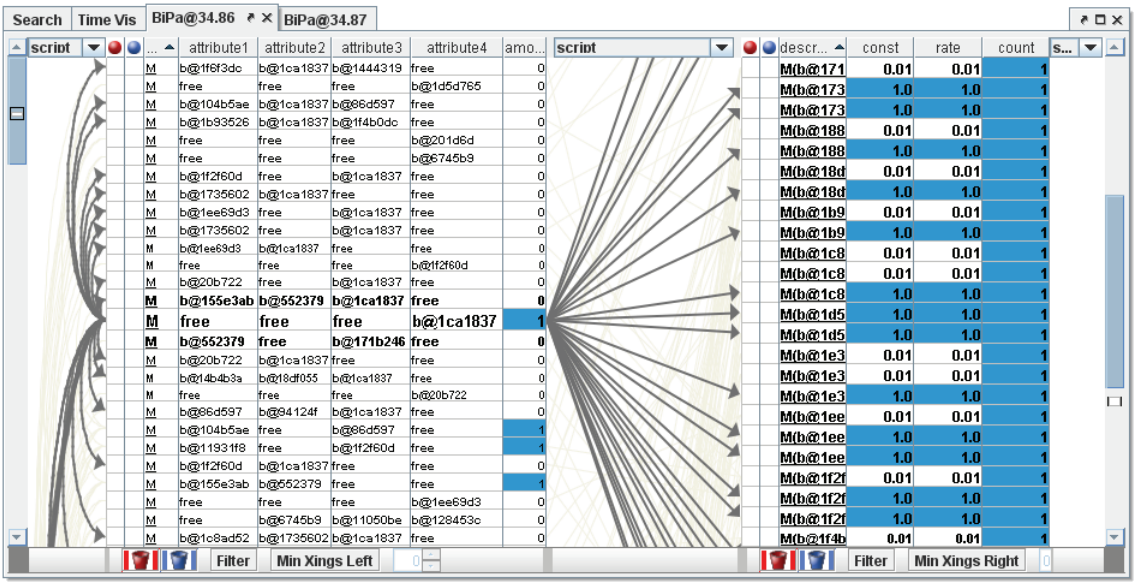
This exemplary exploration walk-through shows how naturally the table-based visualization integrates into a multi-view visualization concept. In a case like above, where the entities (species and reactions) do not have legible names, but just IDs generated by the simulation, the possibilities for sorting by any column and filtering unselected rows make it easy to identify rows with a certain set of attributes or being part of a certain substructure, like a binding reaction waiting to happen as it is shown in Figure 5.24(a).

5.2.4 Additional Remarks

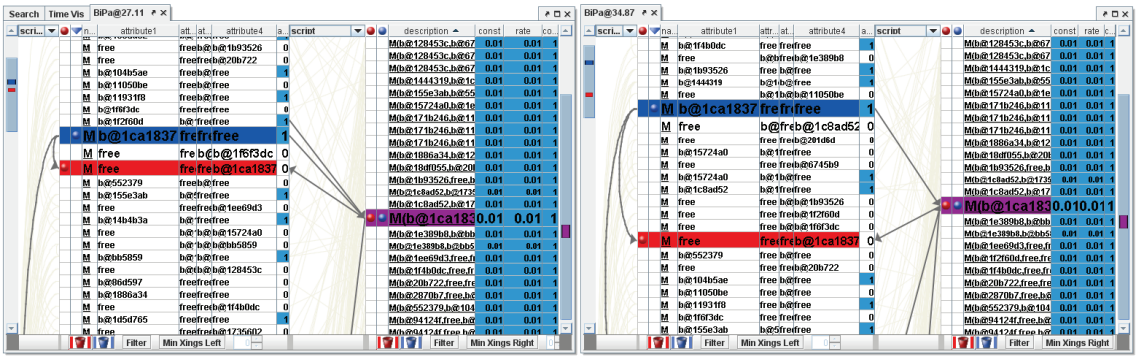
Using the layered layout approach in the form of a tabular representation has proven to be a highly **efficient way of arranging** and rearranging (through sorting) the node sets of bipartite graphs with many node attributes. The table-based representation integrates naturally with a number of the representational adaptations and interaction enhancements, which allow to scale the navigation of the node sets up to multiple 10,000s of nodes. Mainly geared towards the display of node attributes, it also provides a representation of the graph's structure through the display of edges, 1-projections, and the integration of a number of facilities to explore and utilize it, such as edge-based navigation and selection scripts for traversing the graph.

As nodes, edges, and projections have their own respective drawing area, visual cluttering of the layout in one area does not affect the others. So, at no point during the exploration will the nodes and their attributes be occluded by a large number of crisscrossing edges. The two node sets cannot interfere with one another as they also have their respective drawing areas, achieving an **expressive attribute-centric representation** for bipartite graphs, no matter how dense or large the shown graph is.

The application examples have also shown the generality of the visualization approach. It is not tailored to a specific domain and can therefore easily be incorporated in larger multi-view visualization setups alongside other views. The aforementioned advantages of orderability and familarity make it a perfect choice for such an integration. Together with



(a) The actual peak.



(b) Before (*left*) and after (*right*) the peak.

Figure 5.24: Detailed views showing the three time steps before, during, and after the peak from Figure 5.23.

its high degree of interactivity and the possibility to capture complex selection logic in scripts, this allows a multitude of different exploration paths, making it thus an **effective visualization**.

5.3 Summary

Neither of both approaches would have been possible without making use of the inherent structural properties of the respective graph classes:

- In the point-based layout, the tree structure is directly reflected in the hierarchical placement scheme without which the refinement down to a space-filling representation would have never been possible.

- The table-based layout, on the other hand, directly mirrors the bipartite nature of the input graph. Otherwise the compartmentalization in two tabular node displays and edges drawn as lines in between them would not have been possible.

A generic graph layout would have never been able to achieve this for input graphs as large as they were handled in this chapter. Hence, both visualization examples make a strong argument for taking considerations about the input data into account when designing scalable visualizations for them.

The following chapter goes even one step further and argues that not only the specifics of the data must be known to provide adequate visual support for its exploration, but also details about the exploration tasks to be performed on it. Only with this additional knowledge, an adequate visualization support for visual graph exploration can be provided.

Chapter 6

Putting Explorative Graph Visualization in Context

This chapter puts the previous discussion into the broader context of visual graph exploration as a whole by relating the individual levels of data, representation, and task. Such a comprehensive view on all three levels is the last necessary step to support the design of explorative graph visualization that is expressive, efficient, and effective to its fullest extent. Since the representational level alone and the dependencies between data and representation were already discussed in the previous chapters, this chapter sets out to finally integrate and discuss the aspect of exploration tasks.

Single objectives or exploration steps, as for instance "comparison" or "overview", where already considered when designing the visualization techniques from the previous chapters. Yet, exploration does not consist of a single exploration task – the *exploration step*. Instead, it is rather a sequence of exploration tasks – an *exploration workflow*. This workflow encompasses attribute-based and topology-based tasks carried out on different visualization techniques, ranging from overview to detail views, which in turn show different data – e.g., raw data, filtered data, or derived meta-data. This is only logical, as each exploration step can be pursued best with a certain (part of the) data in a certain view. Hence, the task level of explorative graph visualization is actually more of a complex workflow than a collection of individual graph-related exploration tasks that can be subsumed under some preconceived high-level task taxonomy, such as the one given in [LPP+06]. It is important to describe this workflow as a whole, as the effectiveness of a visual graph exploration is not only dependent on the individual exploration steps, but also on their sequential interplay.

Hence, as a first step this chapter aims at describing and structuring the exploration workflow into a conceptual framework that captures individual exploration steps, including necessary preprocessing (data cleaning and filtering) through analytical processing and visualization to the interaction[1]. The framework models the exploration procedure in an abstract way. But, it can be realized as a generic software architecture for explorative

[1] published as "A Framework for Visual Data Mining of Structures" in the Proceedings of the 29th Australasian Computer Science Conference 2006, based in part on the diploma thesis "Visuelles Data Mining komplexer Strukturen" at the University of Rostock 2004

graph visualization. An example of such a software instantiation is also given in the first part of this chapter.

The second part investigates the ties of the exploration workflow to confirmatory analysis. Although confirmatory analysis is not in the focus of this thesis, the benefits for integrating it with exploratory analysis have recently been observed [TK09] and are thus at least exemplary discussed here for the case of a large social network[2]. This part also provides a good example of an edge-centric exploratory analysis, which is an often neglected and underestimated exploration approach.

The last part of the chapter finally brings together all three levels: the graph to be explored on data level, the enormous variety of graph visualizations on representational level, and the exploration workflow of the task level. Towards that end, it introduces a comprehensive description of these levels[3] and integrates them with one another along the lines of the dependencies shown in Figure 3.1. As the sheer number of possible combinations is overwhelming, such a comprehensive description can only be established for a concrete use case or application domain. Within the chosen domain, the model can then be used for a multitude of purposes, for instance, determining a suitable subset of data for a given exploration workflow and guiding the users through the data while keeping their mental map. As a use case for this third part serves the collaborative treatment planning for cancer patients, which integrates graphs in the form of biochemical pathways with a whole set of medical data, such as gene expressions and clinical records.

6.1 A Framework for Visual Graph Exploration

The visual graph exploration workflow is constituted of a number of individual exploration steps and feasible transitions between them. Capturing this workflow is challenging for two reasons: the inseparability of algorithmic and visual analysis steps and the variability of exploration at large.

Integrating algorithmic analysis steps into the visual exploration process can be addressed along the lines of established Visual Data Mining frameworks [Ank01, Kre04] by more or less tightly integrating algorithmics and visualization. Yet, these frameworks have not explicitly been designed for the specific issues raised by the explorative analysis of graph structures. Firstly, this is an issue of efficiency: coupling long-running graph analysis (due to unfavorable runtime complexities) with interactive, visual exploration is not straightforward and a current research subject for visual analysis in general [PTMB09]. Secondly, the uncertain nature of the exploration workflow does not allow to precompute much of the graph analysis in beforehand either, as the concrete analysis steps needed are yet to be determined in the course of the interactive exploration.

This dilemma is inherent in visual graph exploration and cannot be solved without sacrificing either the interactivity by utilizing an alternating exploration scheme that goes

[2] appeared as "HoneyComb: Visual Analysis of Large Scale Social Networks" in Proceedings of the 12th IFIP TC13 Conference on Human-Computer Interaction 2009

[3] appeared in part as "Towards Multi-User Multi-Level Interaction" in Proceedings of the Workshop on Collaborative Visualization on Interactive Surfaces 2009

back and forth between long-running computational analysis and interactive visual analysis, or without losing the flexibility needed for an exploration by predefining a concrete exploration workflow and precomputing the data needed to carry it out. Hence, it is not a solution that is proposed here, but rather an alleviation. It attempts to relax both sides to allow an effective exploration that is still flexible enough to perform a typical graph analysis, but also efficient enough to do as much of it in an interactive, visual manner, as possible. To that end, a two-fold approach is proposed to this problem, each targeting one of the mentioned challenges.

The first consideration concerns the variability of the exploration workflow. While it is in theory possible for each given analysis step to be followed by each other analysis step, this is hardly typical and this degree of flexibility does not necessarily lead to a more effective exploration. For instance, it is common knowledge that the interactive analysis is often performed in a drill-down fashion as described in Shneiderman's mantra (see Chapter 3.2). Under consideration of the mentioned necessary ties to the computational analysis for graphs, this mantra was extended to *Keim's Visual Analytics Mantra* "Analyse first, show the important, zoom, filter and analyse further, details on demand" [KMSZ06]. This structures the exploration workflow in different phases without predefining which exploration steps to use specifically.

This mantra holds as a suitable frame for the second step, targeting the discrepancy between runtime intensive computational graph analysis and interactive visual graph analysis. The idea for this second part of the approach is to gain as much knowledge about the input graph as possible in beforehand and to utilize this knowledge to streamline the exploration workflow. It becomes possible as this knowledge describes the input graph (type, size, density, etc.) and one is now able to select analysis algorithms which are specifically adapted for the described kind of input graph, which are likely to be more efficient – e.g. by using tree algorithms and layouts instead of generic ones for networks. This step of gathering general knowledge about the data up front is already covered by the visual analytics mantra as "analyze first".

Thus, the visual analytics framework for graph exploration presented in this part and shown in Figure 6.1 does not only incorporate the tight coupling of computational and visual analysis in a visual analytics kernel, but also an explicit distinction between:

- **descriptive tasks**, which are basic (yet not necessarily simple) analysis steps that are employed to supplement the raw graph data with additional information that is then available to streamline subsequent exploration, and

- **exploration tasks**, which can be performed as analysis steps in the course of the graph exploration. They are used in a highly flexible manner, allowing them to be undone, further refined, or reparametrized as the exploration progresses.

In the framework from Figure 6.1, both types of tasks are represented by a number of modules with arrows indicating possible transitions from one to another. The tasks and their respective modules, the rationale behind them, and a concrete software architecture realizing the framework are described in the following.

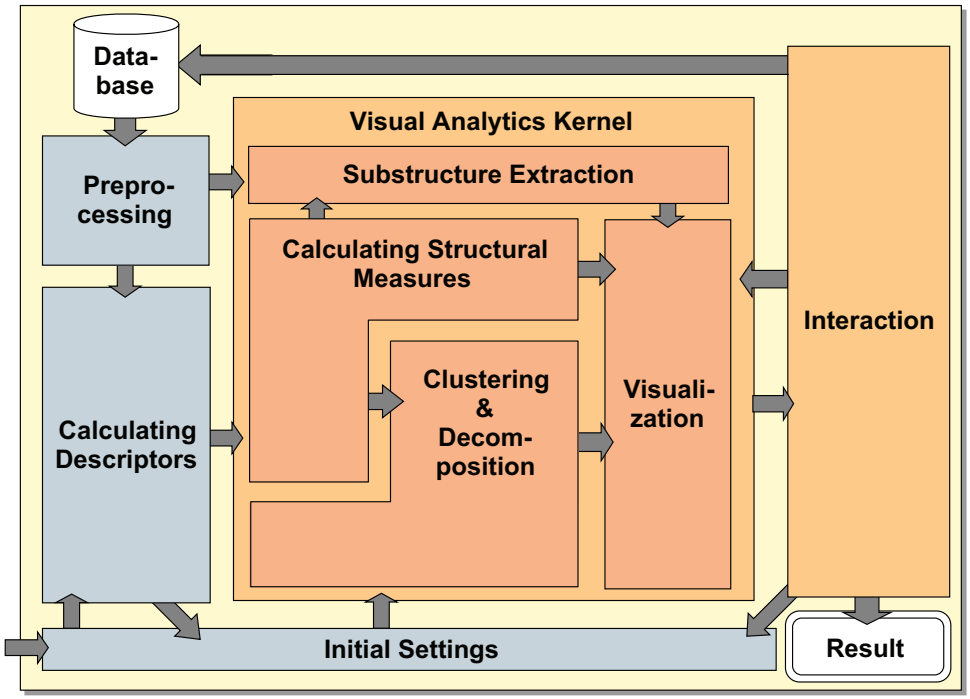


Figure 6.1: A visual analytics framework of graph exploration with modules supporting descriptive tasks shown in blue and modules for exploration tasks shown in orange.

6.1.1 Descriptive Tasks

Descriptive tasks add three different types of knowledge to the graph: common knowledge during preprocessing, user knowledge in the form of initial settings, and derived knowledge from the calculation of descriptors. This is reflected by the three different modules shown in Figure 6.1. In order to decouple these tasks from the exploration, it is important that the added knowledge is a ground truth that is not affected by the insights gained during the exploration process and remains this way, unless the input graph changes. This separates descriptive tasks from exploration tasks – for instance, in most cases the result of a clustering algorithm cannot be precomputed, as it often needs interactive reparametrization throughout the exploration to provide meaningful clustering results for each exploration step. In short, the three different kinds of descriptive tasks are:

preprocessing, which includes mainly procedures for data cleaning, thus ensuring a well defined graph to start the exploration with. Hence, these steps add *common knowledge* about the data type and filter out parts of the data that do not conform with the commonly agreed upon notion of graphs, for instance dangling edges with only one defined endpoint and the other end being "loose".

initial settings, which question the user for additional information about the graph (e.g., whether to interpret its edges as directed) and the exploration workflow as far as it is already planned (e.g., available time frame). It is this part of the framework, where *user knowledge* can be asked for to be integrated in the later exploration.

calculating descriptors basically means to compute meta-data about the graph by merging the cleaned data and the information given by the user, e.g., if the graph was determined a directed graph during initial interaction, the preprocessed data can be checked automatically if it also fulfills the criteria of being a directed acyclic graph. It is this step that adds automatically *derived knowledge* to the data.

Entering a thus enriched data set in the modules of the following explorative part, does not only guarantee a well-defined input graph, but it also carries additional information about itself that lead to a more efficient algorithmic and visual analysis as more appropriate analysis procedures (specific algorithms/visualizations) can thus be chosen.

6.1.2 Exploration Tasks

As in exploratory analysis in general, the back and forth of an interactively steered graph analysis stands at the center of the framework in the form of the *visual analytics kernel* and the *interaction*. Both encapsulate the exploration steps as performed by the machine (algorithmical steps of the analytical kernel) and by the analyst (visual steps through interaction). Intermediary results, such as subgraphs of interest selected in the course of the exploration, can be stored for later in-depth exploration (arrow back to the database in Figure 6.1), effectively branching the exploration workflow.

As mentioned, it is often hard to separate between algorithmical and visual steps, as both are intrinsically tied: visualizations need a certain degree of precomputing (for instance, a hierarchical clustering for an adaptive level of detail as described in Chapter 3.2) and computations need a graphical output of their results. This tight connection between both aspects is incorporated in the visual analytics kernel, which is in turn steered and parametrized via the interaction module.

Additional reoccurring patterns of the exploration workflow can be incorporated as submodules, as it was done in the visual analytics kernel. For instance, a computation of structural measures (e.g., similarity or distance measures) is usually done before a graph clustering is triggered, as the clustering needs the distance measures as an input. Known dependencies like this can hence be added on top of the framework.

6.1.3 Realizing the Framework as a Software Architecture

The presented visual analytics framework captures the process knowledge about visual graph exploration on an abstract level. Once captured, this process knowledge can now be distilled into a concrete software architecture that is thus best-suited to support visual graph exploration. The idea behind such a straight-forward implementation of the framework is to perceive it as a “pipeline” for graph data with additional meta information through the different modules realizing the exploration. These modules are then populated with concrete analysis algorithms, visualizations, and interaction mechanisms that are required for a given setup or domain.

Figure 6.2 shows an example for a number of possible analysis algorithms, visualizations, and interaction mechanisms as parts of the framework implementation. The individual exploration workflow is then specified as a path through such a concrete frame-

work, progressing from one module to the next. Hence, an implementation of such a pipeline results automatically in a plug-in architecture that is able to incorporate a diverse set of algorithms and visualizations, and ties them together along the lines of the pipeline. To show the feasibility of such an implementation, a proof-of-concept prototype has been realized. A screenshot of this prototype is shown in Figure 6.3. It features a selection of graph layout techniques (e.g., the Fruchterman-Reingold's spring embedder method [FR91] for networks and the MagicEye View [KS02] for trees).

Noteworthy is in this context the novel concept of *tree-likeness*, a descriptive task module that quantifies how similar to a tree a given graph is. This is done by precomputing the tree-likeness as a tuple (p,k) with k being the absolute number of cross edges and p being the relative number of cross edges with respect to all edges. In the course of the following exploration, this additional information about the graph helps to determine whether

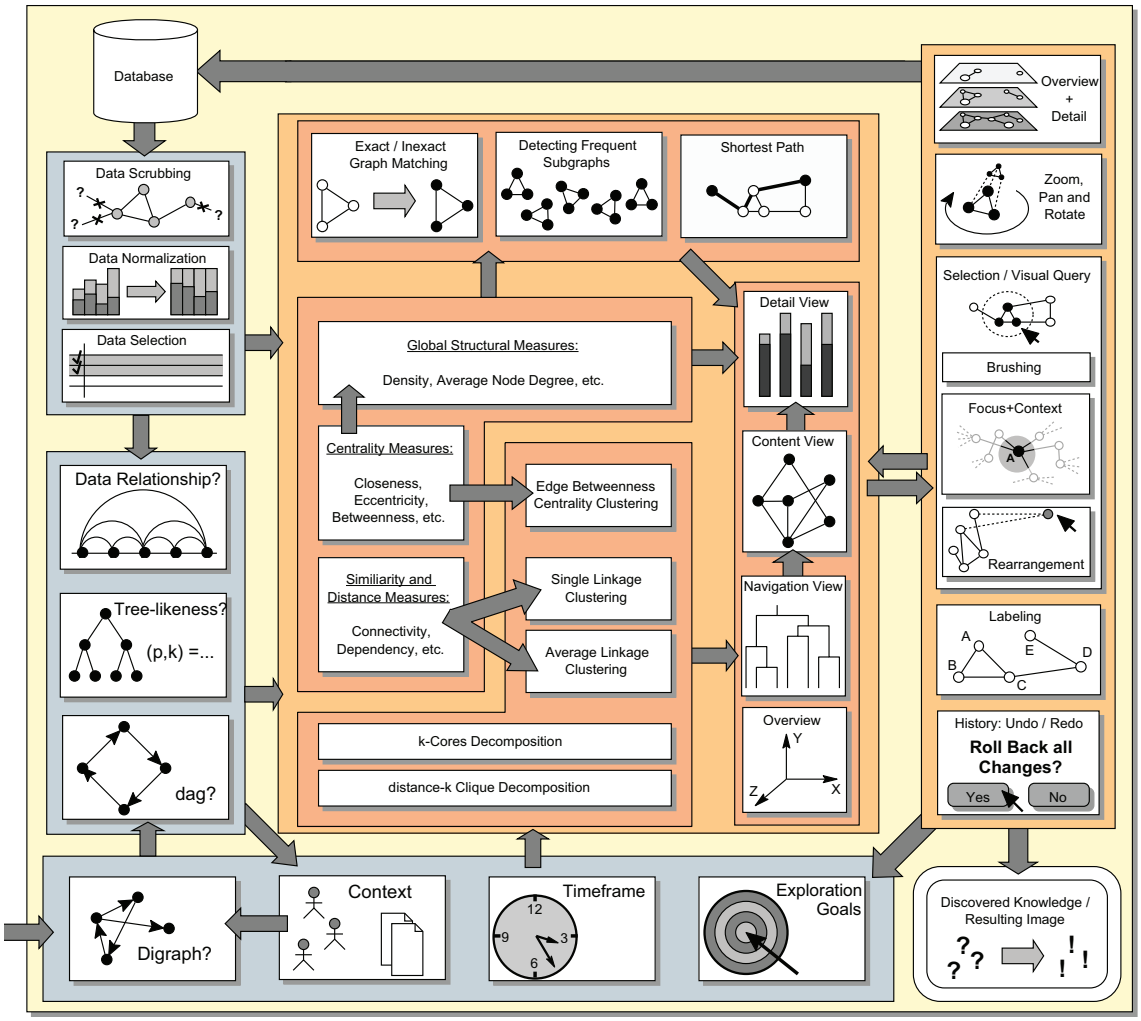


Figure 6.2: The framework from Figure 6.1 with concrete analysis algorithms, visualizations, and interaction mechanisms plugged into the different modules.

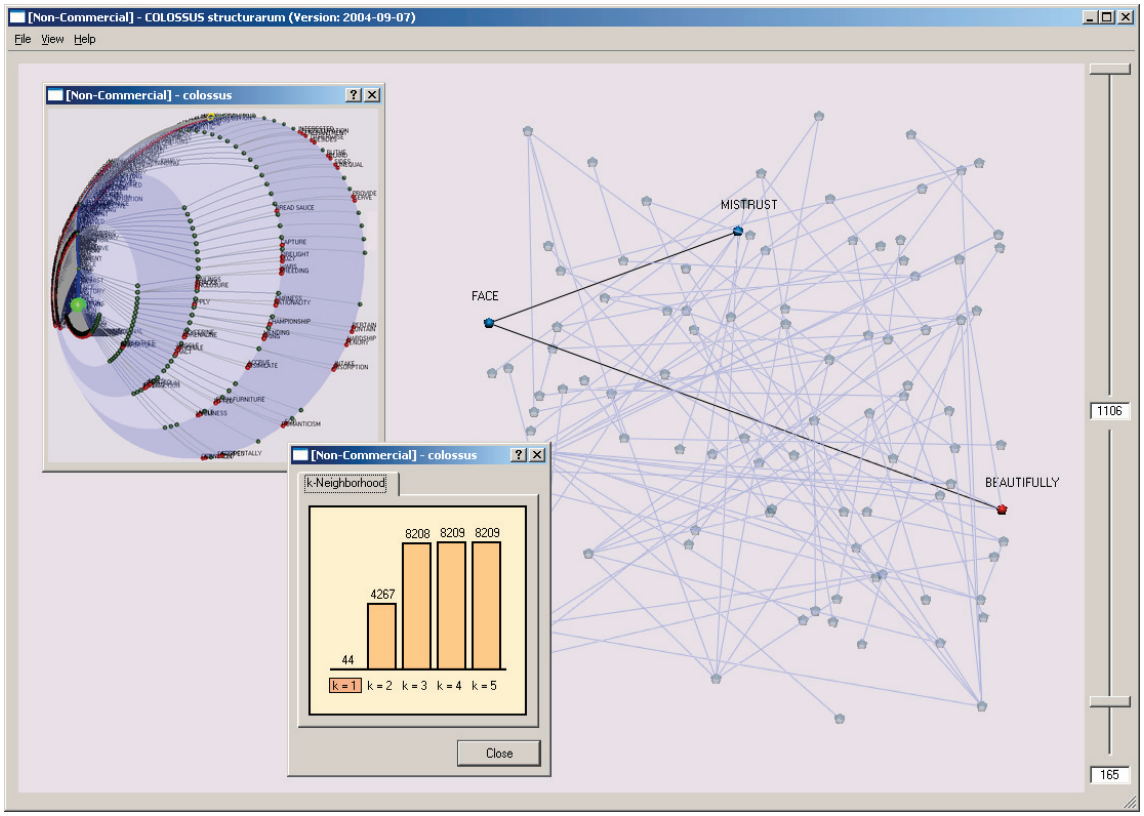


Figure 6.3: A screenshot of the framework's prototypical implementation. The view in the background shows a subgraph of the larger Edinburgh Associative Thesaurus data set (http://www.eat.rl.ac.uk/) [KAMP73]. It has been filtered using the filter sliders on the right, and it depicts a highlighted shortest path and a red colored, selected node. The navigational tree view in the top left contains the browsable result of a hierarchical cluster algorithm. The bar chart shows a detailed view of the neighborhood distribution of the selected node with its immediate neighborhood being selected. All three views are linked, so that a selection in one view will be relayed to the other views.

to use a network layout or a tree layout with cross edges, as described in Chapter 3.2. This is a useful consideration, as especially Chapter 2.4.1 and Chapter 4 have illustrated the multitude of available tree visualizations as well as their benefits (e.g., a low runtime complexity for their layout), which are thus made available to tree-like graphs, too. The combination of an absolute and a relative value is necessary, as tree layouts can only handle cross edges if their number does not exceed a certain layout-dependent threshold and these extra edges being the exception and not the rule. Both thresholds – the one for the absolute and the one for the relative number of cross edges – can be adjusted by the user as part of the initial settings.

As the workflow of explorative graph visualization has hereby been described and implemented, the following part discusses its ties to confirmative graph analysis using a concrete application example from social network analysis.

6.2 Confirmatory and Exploratory Graph Visualization

Utilizing the introduced framework or a software implementation of it results in a new observation (hypothesis) about the graph, if applied successfully. That is what exploratory graph visualization is about: visually supporting the observation of previously unknown characteristics of the graph – be it the homogeneity of leaf attributes in implicit tree visualizations, balancing issues in the point-based tree visualization, or bottlenecks in the table-based visualization of bipartite graphs. It thus covers the aspect of "discovering the unexpected" [TC05] of a visual analytics workflow. Yet, its second aspect of "detecting the expected" is no more part of an exploratory, but rather of a confirmatory analysis as it confirms whether preconceived graph characteristics are indeed met. For instance, this can be characteristics inferred from the graph's application domain or newly observed characteristics whose significance is yet unclear. This part aims at bringing both aspects together for a concrete application example. Its background, the concrete data, the tasks to be performed on it, and the visual representation to be used are detailed in the following. The three actual analysis steps of confirmatory analysis, exploratory analysis, and a combination of both are given afterwards, including some findings to illustrate the usefulness of the analysis steps taken.

6.2.1 The Overall Setup of the Application Example

The example used in this part of the chapter to point out the benefits of combining exploration and confirmation in the tight sense of visual analytics is taken from the field of social network analysis. Its different aspects as they apply to this example are shortly outlined in the following.

Visual Exploration in Social Network Analysis

The analysis of social networks has had a major influence on social sciences ever since it was first proposed [Fre04]. Besides metrics and algorithms to analyze social networks [WF94], visualization plays an important role in this field. From the beginning, *sociograms* and *sociomatrices* were used as representations of the analyzed social networks. Sociomatrices cross-tabulate a particular connection metric over a number of actors (nodes on the network) and sociograms are traditional node-link representations of the network. The node-link representation of sociograms has been echoed in many social network visualization tools, for instance the advanced Perer and Shneiderman's *SocialAction* [PS08], which allows users a fully interactive, visual examination of social networks in combination with different statistical analysis methods. Sociomatrices, being similar to adjacency matrices, have been advocated for medium scale social network visualization by Henry and Fekete [HF07] using a hybrid representation of a matrix with overlaid links. Both approaches have their specific advantages, as it was described in Chapter 2.2.

The Data Set

The example discussed here uses a data set taken from a social network site running internally at a large multinational company. The site is an opt-in site that people use for connecting with other employees, through the sharing of photos, lists, and events [DMG+08]. Part of the process of connecting is to directly link to other employees, as is typical on any social network site. When a user on the site adds someone as a connection, the other user is not required to reciprocate the connection, so employees can connect to anyone inside the company, without the need for the other person to join the site first or reciprocate back afterwards. The data set used as a sample is a snapshot that was taken mid-July 2008. At that time, 37,000 employees had joined the site and they had formed approximately 300,000 connections.

The Analysis Tasks

Typically, social network analysts use a combination of algorithmics and visualizations to perform a rather node-centric analysis by determining central actors, dense clusters of actors, or degrees of separation between actors in relatively small networks of at most a few hundred nodes [dNMB05]. For larger networks, the analysis often resorts to small ego networks that capture only the neighborhood of selected actors, but leave out the more distant network regions. Yet, from an organizational perspective, it is rarely the individual actor that is of interest at that level of scale, but rather overall connection patterns within the company. Hence, the (accumulated) connections between (aggregated) nodes come into focus. For the intra-organizational network used here, it may be more interesting to investigate properties of the connections instead of properties of the actors – for instance, how the Asian branch of the company connects out to its legal department, instead of which one of the 37,000 actively participating employees has the most connections.

To generate such an abstracted, top-down view, different aggregation hierarchies can be employed for the actors – e.g., the management hierarchy to correlate connection behavior with organizational structure or the geographical hierarchy based on the user's working location (i.e. continent, country, state, city, building, floor, office) to correlate connection behavior with geographical location. In addition to the hierarchization, with ca. 300,000 connections in the sample data set, automatic means of determining potentially interesting or anomalous connections are absolutely necessary. Otherwise, the user can spend a significant amount of time browsing the network at multiple levels of scale without learning anything new. These automatic means can range from simple metrics to intricate graph algorithms. Yet, in the light of the size of the data set, this example focuses on fast to compute metrics. An overview on viable edge metrics that can be computed towards that end is given in [MS08].

The Visual Representation

In the context of this case study, a matrix representation is used, as it is a very edge-centric representation, which fits an edge-centric analysis well, and grants equal space to existing and missing connections – an important point, when it comes to visualizing expected yet

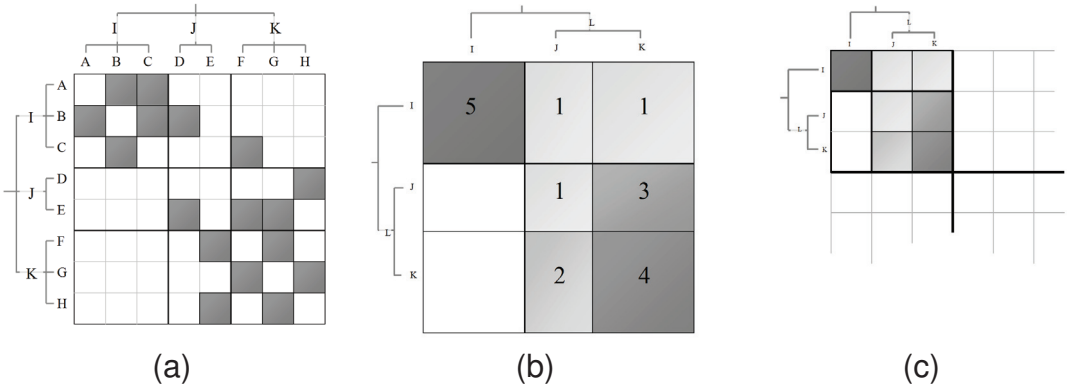


Figure 6.4: Collapsing an 8×8 adjacency matrix to a smaller 3×3 matrix: (a) original matrix with node hierarchy on both sides, (b) collapsed version of the matrix with lowest level of the hierarchy eliminated and edge counts aggregated, (c) this collapsed version itself forms a small section of a higher level adjacency matrix.

not present connections during confirmatory analysis. In combination with a hierarchical grouping along the lines of the geographical or managerial hierarchy described above, it is able to handle a few million connections with ease and thus scales well beyond node-link representations [vH03, AvH04, EDG+08]. This grouping reduces the size of the $37,000 \times 37,000$ adjacency matrix by collapsing the matrix cells and aggregating the number of connections. Figure 6.4 shows how an 8×8 matrix can be collapsed to a 3×3 matrix by using a predefined hierarchy to aggregate cells. The resulting collapsed matrix color codes the total number of edges for each submatrix below (darker colors indicating more edges) and still maintains many of the features of the original network. For example from Figure 6.4(b) it can still easily be seen that there are no connections from groups J and K to group I and that there are relatively many connection among nodes in group I. As a final step, the cell sizes are normalized so that the same process can be recursively repeated with the collapsed matrix. Note that, even though the relative difference in cell sizes in Figure 6.4(b) accurately portrays the difference in the number of leaf nodes, adjacency matrices with irregular cell sizes are much harder to comprehend as they grow larger, especially when the difference in cell sizes is large.

Besides using the hierarchies to ensure the visual scalability of the matrix representation and to guide the drill-down into the data, it is also used to achieve computational scalability by enabling to handle the network data in a semi-external memory approach. This means, that the entire nodeset and the hierarchy of the network are kept in RAM while a relational database stores the actual connections between the nodes in the network. When a higher level view of the network is requested, aggregation of edges in the database is done on the fly using a fast lookup algorithm as shown in Figure 6.5. This way, it is possible to explore even large graphs with a reasonable amount of memory – graphs with 5 million edges use about 200 MByte of RAM.

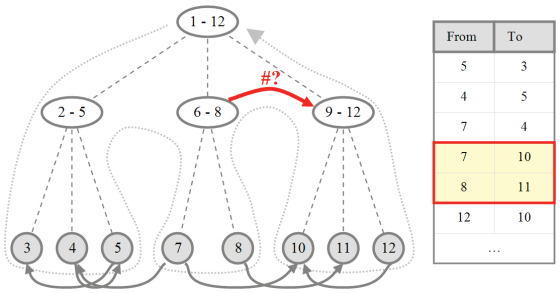


Figure 6.5: Schematic representation of the network (dark grey nodes and curved connections) and the aggregation hierarchy. By numbering the nodes in the aggregation hierarchy in a depth first manner (dotted line) and keeping track of the minimum and maximum values encountered during this traversal the number of edges connecting groups (6-8) and (9-12) can be determined by running the query: SELECT COUNT * FROM EDGES WHERE 6 ≤ 'From' ≤ 8 AND 9 ≤ 'To' ≤ 12;

6.2.2 The Analysis Process

The following three analysis steps all combine the computation of an edge metric with a subsequent visual analysis. It follows Keim's mantra in the sense, that the edge metric computation is a part of the "analyze first" step, whereas the rest of the visual analytics workflow is directly supported by the matrix visualization, e.g., zooming in by stepping down the aggregation hierarchy.

As mentioned, in this example the analysis consists of three steps: at first, a straightforward confirmatory analysis is carried out by aggregating the number of connections between nodes, visually exploring them and confirming background knowledge about a typical social network characteristic. In a second step, explorative analysis is used to check the discrepancies between ingoing and outgoing connections, uncovering new insights about asymmetrical connection behavior in the data. As a last step, a novel edge metric – the deviation from the expected – is applied to gain additional insights via exploration and at the same time, get visual feedback on the significance of these findings via statistical confirmation. The latter metric combines explorative and confirmative analysis in the way it is envisioned in for visual analytics [TK09].

Step 1: Connection Count (Confirmatory)

The number of aggregated edges, or *connection count*, is probably the most straightforward metric one can think of. It basically captures the strength of connections between (aggregated) actors. Since it is a simple summation, e.g., of all connections running from employees of department A to employees of department B, it is traceable, easy to interpret and to implement, and has been used successfully in previously published edge-centric network analyses [vH03, AvH04]. The connection count CC for an edge (X,Y) can be expressed as:

$$CC(X,Y) = \|(x,y) \in E : x \in desc(X) \wedge y \in desc(Y)\|$$

with $desc(X)$ indicating the set of descendant leaves of a node in the hierarchy. To be able to compute connection counts between groups efficiently a special numbering scheme is employed on the nodes in the network. The hierarchy on the nodes in the network is traversed in a depth first traversal starting at the root, and the encountered nodes are incrementally numbered. For each node in the hierarchy the minimum and maximum values encountered for that node are kept track of. These Depth First Search numbers can then be used to query the edgelist stored in a database whenever the total number of connections between two groups is needed (see Figure 6.5 for details). Combined with caching of the top level matrices this keeps the analysis memory-efficient and fast. Alternatively, a solution where all edge counts are precomputed and stored in a disk-based index can be used [EDG+08], although this makes it harder to switch between multiple hierarchies.

Color-coding the connection count appropriately onto the matrix cells, allows it to support any task that can be translated into a suitable color scale or color mapping function [TFS08]. This includes the comparison, localization, and identification of specific *CC*-values. Figure 6.7(c) employs for example a logarithmic white-to-red color scale. The logarithmic mapping is used to alleviate the problem of the large number of self-connections on the diagonal drowning out subtler off-diagonal patterns.

Yet, the high values on the diagonal do confirm an anticipated connection pattern for social networks, as actors usually tend to connect to their own folks – either their countrymen when using the geographical hierarchy or their immediate coworkers when using the managerial hierarchy of the company. When zooming in, this pattern persists also on lower levels, such as state-level or town-level. If this pattern was not present, it would have been worthwhile to investigate the cause. Yet, since this is not the case, the following exploratory analysis step should reveal additional insight.

Step 2: Asymmetry (Exploratory)

In this exploratory analysis step, asymmetry in connection behavior shall be investigated, since connection behavior does not need to be reciprocated (i.e. X can connect to Y, but Y does not have to connect back to X). As connection behavior is often set equal to communication behavior, such imbalances can become very harmful to a company's productivity, if they occur on a higher organizational level and affect entire departments. Asymmetry metrics can also be useful in wider contexts, especially if the connections' weights can be significantly different for each direction. Examples of such networks include trade and financial networks. Asymmetry can be defined for each edge (X,Y) as:

$$ASYM(X,Y) = CC(X,Y)/CC(Y,X)$$

As it was shown that the computation of *CC* can be done very efficiently, *ASYM*'s computation is equally efficient, as it depends solely on *CC*-values.

It is color-coded to the matrix visualization using a double-ended color scale ranging from blue (lowest value – more edges coming in than reaching out) through white (1 – symmetrical) to red (largest value – more edges reaching out than coming in). Small imbalances in asymmetries can distract from detecting large imbalances, so the asymmetry values are plotted only above and below a small cutoff interval around 1.

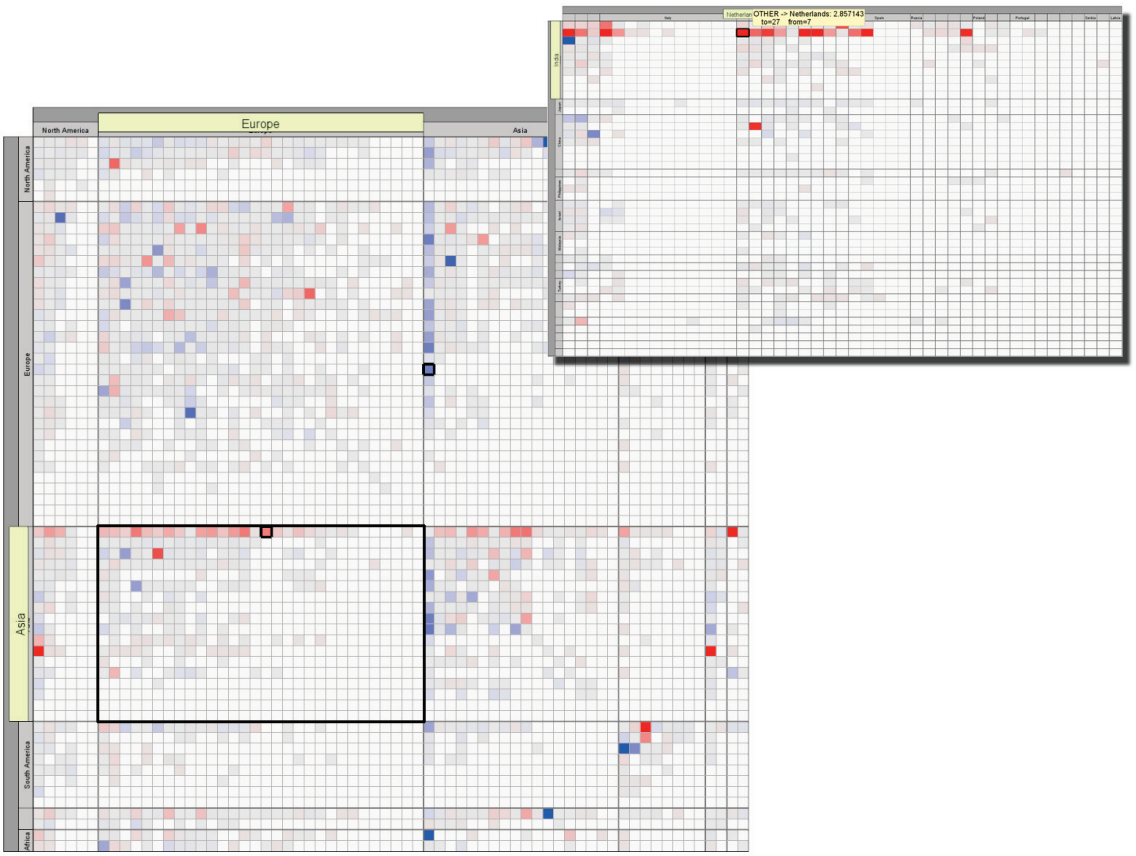


Figure 6.6: A matrix view colored according to the computed asymmetry, the difference between incoming and outgoing connections, with India showing up as an outlier. The horizontal red band (positive asymmetry) and the vertical blue band (negative asymmetry) indicate there are many unreciprocated connections from Indian employees. The inset shows a zoomed view of the outlined cell with the connections between Asia and Europe, the red horizontal band represents connections from mobile employees (with location OTHER) in India to people in Europe.

As Figure 6.6 shows, using the geographical hierarchy to aggregate, interesting and previously unknown patterns emerge immediately for this example: a large horizontal (red) and vertical (blue) bar appears in the row and column representing India. This indicates India overall has more connections going out to other countries than it has coming back. The team that had developed the social networking application was aware of the fact that a lot of US users were getting friend requests from users in India they did not know personally, but did not know that this pattern persisted over different countries. To determine where this behavior was coming from, one can drill-down further and explore the connections from India to Europe (inset). It can be observed that most of the people responsible for these asymmetric connections report their work location as ‘OTHER’, which in practice means a non-traditional office, such as drop-in stations or home office. Hence, these people may be using the social network site with a particular focus on meet-

ing new people, because they do not have regular face-to-face contact with coworkers.

In a separate survey, 2,000 users of this social network service were asked about different reasons why they were motivated to use the site. Comparing the responses between countries, users in India reported to a significantly higher degree than other countries that they were using the site for getting to know people they would not otherwise meet at the company, using the site to find experts and using the site to discover people with similar interests. All three of these activities involve reaching out and connecting across organizational and cultural boundaries, and these motivations offer a partial explanation of why this particular cluster of outward links may exist.

So, the exploration generated new insight, which was confirmed by the developers of the social network site and explained through a user study. Including the confirmation directly with the exploration is the next step of the analysis.

Step 3: Deviation from Expected (Combined)

The main problem with the earlier metrics is to judge whether an observed pattern is significant. This is especially important when management decisions depend on the analysis. In the last analysis step, significance was confirmed by outside experts. It is the aim of the edge metric introduced in the following to combine support for the explorative aspect of gaining new insights, as well as support for the confirmative aspect of rating its significance. One way of estimating the significance of an observation is to try and determine what part of this observation might be explained by pure chance. In other words, if the edges E in the graph were distributed completely arbitrarily, how many edges can one expect to fall between Europe and India[4]? Note that the total number of connections in a row X of M is equal to the total number of edges X_{out} that have their startpoint in X, and the total number of connections in a column Y of M is equal to the number of edges Y_{in} that have their endpoint in Y. The probability of having an edge connecting X and Y if the data were randomly distributed (i.e. the choice of start and endpoint of an edge are independent) is then equal to $P(X,Y) = \frac{X_{out}}{E} * \frac{Y_{in}}{E}$ with an expectation and variance of:

$$EXP(X,Y) = E * P(X,Y) = \frac{X_{out} \cdot Y_{in}}{E}$$

$$VAR(X,Y) = EXP(X,Y) * (1 - P(X,Y)) * \frac{R * C - E}{R * C - 1}$$

Here, the variables R and C denote the numbers of rows and columns of the part of the matrix variance and expectancy are related to. Typically $R * C = N^2$ if variance and expectancy are to be computed with respect to the distribution of the edges over the whole graph. But one can also determine the expected values given the characteristics of a particular subcell (typically an ancestor of the cell one is looking at) in the matrix, e.g., computing the expectancy of the connection behavior of an Asian country not with respect to the edge distribution over the whole world, but only with respect to Asia.

[4]This problem is similar to a chi-square analysis of a set of observations, using the matrix M of connection counts as the contingency matrix.

The difference between the observed number of connections and the number of connections one would statistically expect if the distribution of the edges were random is mapped onto the matrix representation using the same double-ended color scale as before: red cells indicate a higher number of connections than expected, blue cells indicate a less than expected number of connections. Additionally, the size of the glyph in each cell indicates the significance of the deviation, which is determined by relating the observed deviation to the computed variance that is likely to occur just by chance. Hence, if the square indicating the deviation from the expected has shrunk to a single point, it is (statistically) not significant – yet, if it fills the entire matrix cell, it is very much so.

A good test case for the deviation metric is to see if the company's business units correspond to their respective geographies. This should be the case, as it was mentioned in Step 1, that self-connections are frequent in social networks. Hence, as an example, one can expect employees of the German business unit to have most of their contacts with employees in Germany. Other business units span different geographies however, and the deviation metric should also point this out. Thus, Figure 6.7(a) shows a matrix display where the outgoing nodes have been grouped using the organizational hierarchy and the incoming nodes using the geographical hierarchy. The expected patterns show up very clearly, e.g., for the German and the Britain division, thus confirming the suspected dependency.

The same metric is also applied to look at connections within Europe, shown in Figure 6.7(b). It allows to detect a number of interesting patterns. Again, the tendency to connect to people in the same country is fairly obvious. Other observed patterns have to do with language and geographical proximity (for example, Switzerland connects more than expected to Germany, Austria, and France) or with organizational structures (Sweden, Norway, Finland, and Denmark are all part of a single organizational structure). Another pattern showing up is the larger than expected number of connections between Austria and many of the countries in Eastern Europe. This observation is equally valid, as the company, like many other multinational firms, handles a significant part of its Eastern European business from Vienna, which explains these larger than expected connection values.

The latter observation even persists across different data sets: the company also uses internal online document sharing services where people can upload a single document and share it with multiple colleagues, which saves the user from having to mass-mail the document. This second data set contains 75,000 document sharing relationships (edges), where two actors are related if one of them has shared a document with the other. Figure 6.7(d) shows document sharing relationships over different countries in Europe. Again, Austria appears to connect significantly more to Eastern Europe than other countries. The asymmetry is expected since sharing a file is an activity that is usually not reciprocated by the receiving user. Still, the rough similarities between the images in Figure 6.7(b) and 6.7(d) verify that both patterns revealed by the tight integration of exploration and confirmation reflect meaningful organizational patterns in the company.

Overall, this three-step example shows how beneficial the integration of analytical measures and visualization, as well as exploration and confirmation can be. Without the

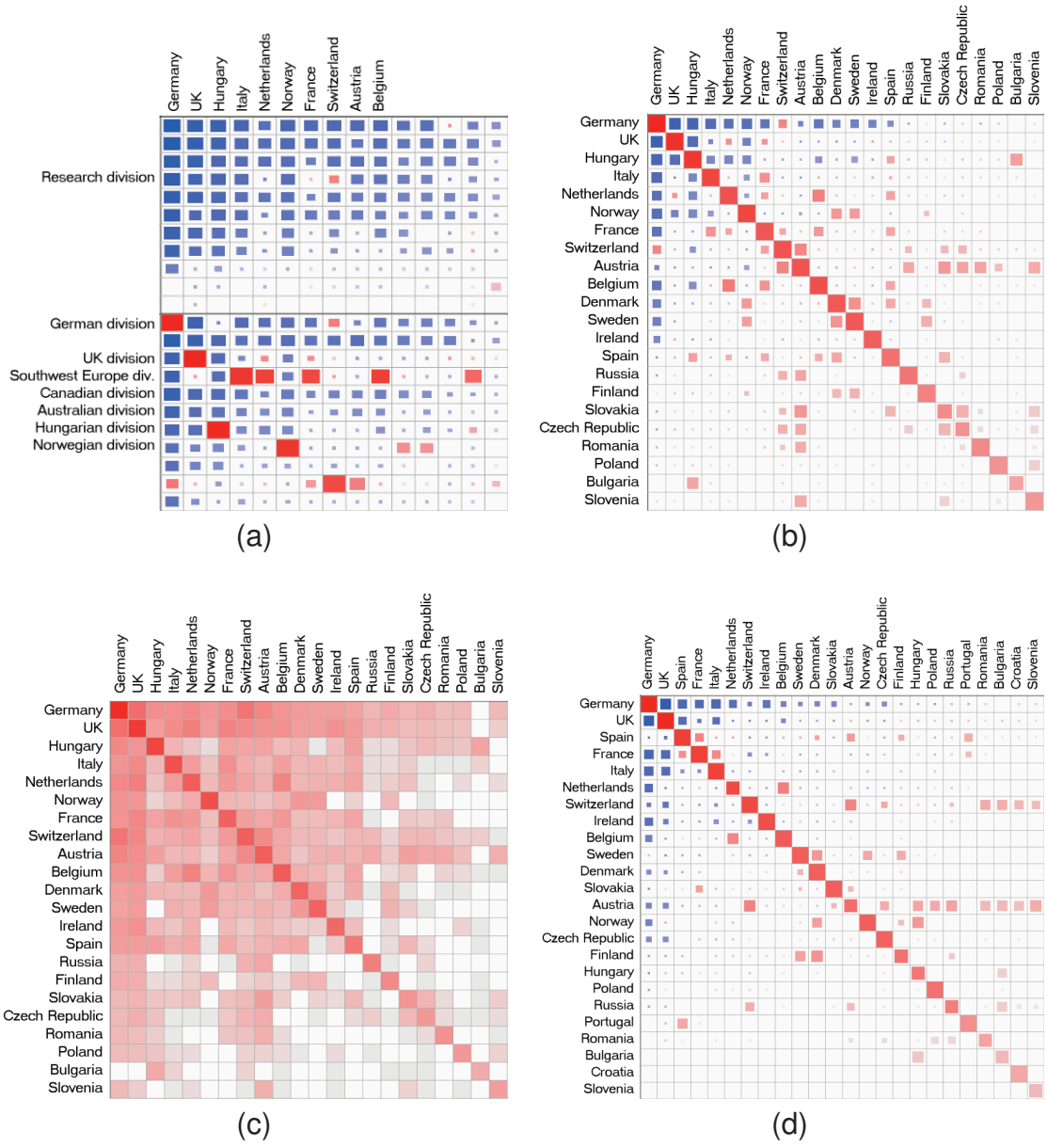


Figure 6.7: The deviation from the expected number of connections between divisions and countries (a) and between connections among different European countries (b). For comparison, (c) shows the same view as (b) but using the connection count metric, and (d) shows the same part of the matrix as (b) and (c), yet for a file sharing network within the same company.

computational analysis and subsequent visualization, a user could hardly interactively explore a network of this size for unknown patterns and judge their significance at the same time. It is also powerful in that regard, as it brings out previously hidden patterns, such as the regional or language dependencies. The following third part of the chapter carries the idea of integration even further – different data sets, different views, and different tasks

are combined into one comprehensive model, which is in turn used to automatically derive suitable data/view combinations for stepping seamlessly through an analysis workflow, be it exploratory or confirmatory.

6.3 Exploration of Multiple Data Sets Following a Predefined Workflow

So far, this thesis has focused on supporting the exploration of often large, yet isolated data sets. But, the example from the last part already showed how beneficial it is to bring in further data – e.g., the second data set of file sharing activity to substantiate a pattern found in the social network. To handle multiple data sets is a frequent requirement in many applications, which has been reflected upon in visualization research, e.g., as a flexible visual fusion approach for multiple parameters in [WFK+02]. Approaches like this mainly concern the adaptation of the representation to the changed conditions on data level in order to achieve a given task. This *forward direction* of thinking from the data to the task is certainly predominant in the field of information visualization and also in this thesis: the data characteristics (graph size, type, and other descriptors such as tree-likeness) stand always at the basis of visualization design. This is well justified, as the single data set stands as given with no option of choosing otherwise. But in a scenario with multiple data sets, there is a chance to select the most appropriate one for a given task, which thus corresponds to the reciprocal approach following a *backward direction* from the task to the data. Including this backward direction into the visualization design process allows a holistic design approach that considers the interplay between data and task with the final representation from both sides.

This principle is illustrated in Figure 6.8, with continuous arcs representing the dependencies of visual analysis (as used for the discussion above) and dashed arcs representing those of direct computational analysis (as it is needed in conjunction with the visual analysis to realize visual analytics). This principle is explicitly not restricted to single tasks (analysis steps), but also applicable for entire task sequences (analysis workflows)[5]. In this case, it is not only the question which data set to choose with respect to a single analysis step, but also to choose it under consideration of the previous and next analysis steps. If possible, the choice of an adequate data set for each analysis step should ensure a *seamless transition* from one step to the next by maintaining the mental map of the analyst. This can for example be done by utilizing dependencies between data sets – e.g., one data set B describing the properties of an item from another data set A. Such dependencies (e.g., part/whole or equivalence relations) bridge the gap between different data sets as they provide a semantic connection between them, which can be exploited for a coherent analysis. Switching back and forth between such data sets can either be seen as a drill-down analysis ($A \rightarrow B$: exploring its detailed properties in B after selecting a data item in A) or as a comparison to put details into context ($B \rightarrow A$: exploring items

[5] After having made the point that a fruitful exploratory analysis integrates elements of confirmation as well, and the model presented in this last part being applicable to both types of analysis, here the more general term "analysis" is used instead of "exploration."

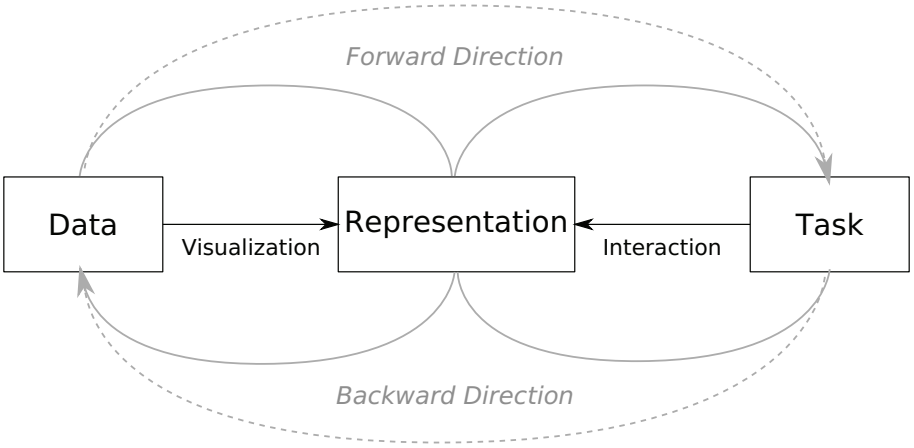


Figure 6.8: Design directions for a comprehensive visual analysis: the typical forward direction from the data to its representation and finally the task to be performed, and the introduced backward direction from the task to the representation used for it and finally the most suitable data set to perform the task on.

in *A* with similar properties as the ones investigated in *B*). A typical analysis workflow, such as Keim's or Shneiderman's mantra, can make use of such dependencies between data sets and extend themselves across the boundaries of a single data set. For instance, at a certain level of the "Zoom and Filter" step on data set *A*, a seamless transition to the next data set *B* can be performed to advance smoothly and without losing the mental map to the following "Details on Demand" analysis step.

As these dependencies are an inherent property of the different data sets, it implies that analysis workflows like Shneiderman's mantra are not necessarily restricted to the view level, as it is typically perceived. Instead, these workflows can be extended directly to the data level as well[6], if dependencies between data sets exist. A task-driven visualization design can thus decide whether an analysis step is better performed visually using the graphical representation or computationally using analysis algorithms or simple SQL queries and combine them as needed, providing true visual analytics in this regard.

A task-driven, comprehensive design, focuses mainly on the selection of appropriate and seamless data sets for each analysis step, as well as suitable visualizations for visual analysis steps. In order to provide the means necessary for such a design, methods to describe the interdependencies of the aspects involved (data dependencies, view dependencies, task sequences) are introduced and subsequently combined into a comprehensive model, describing the given analysis scenario. The resulting model captures all feasible visual analysis paths and thus the inherent variability of an analysis in the form of a network connecting the three levels. The model itself helps to ensure expressiveness and efficiency by relating only such views to data sources that display them truthfully and fast, and effectiveness by integrating only such views with tasks that actually help them along. The analysis process is then perceived as a traversal or navigation of this compre-

[6]as first suggested in "Visualizing Graphs – A Generalized View" in the Proceedings of the 10th International Conference Information Visualisation 2006

hensive model or a subset thereof, transitioning from view to view, from data set to data set, and between both levels. At each step of the analysis, it allows to determine among all possible analysis steps (transition to any data set and view) the feasible ones and guide the analysis in their direction.

In the following, the general and use case independent fundamentals of building these comprehensive models are given for all three levels. Afterwards a concrete, biomedical scenario is used to exemplify this concept with a real-world clinical task. This example scenario was conceived in close collaboration with the Institute for Computer Graphics and Vision of the TU Graz and their Caleydo[7] biomedical visualization project team.

6.3.1 Conceptual Foundations

While this concept surely surpasses the comparatively simple analysis workflows devised previously, it still builds upon the fundamental drill-down analysis as formulated by Shneiderman and extended by Keim. Their mantras define the fundamental tasks of overview/showing the important, filter, and details-on-demand, and it is common knowledge that most visualizations are most effective for one of these tasks. For instance, the point-based tree visualization from Chapter 5.1 has been specifically designed to serve mainly as an overview visualization and its filtering functionality is far from being as sophisticated as the one provided by the table-based visualization for bipartite graphs in Chapter 5.2.

In order to make that connection between a visualization technique and the data/task they are suitable for, a description of data, views, and tasks is needed. Once authored by a domain expert, it can then be utilized for the subsequent task-driven design. While authoring these descriptions is certainly time-consuming, once they are prepared they can then be used as often as needed to automatically derive custom subsets of the available data and views to specifically support given analysis workflows. The domain knowledge needed for the three individual descriptions is:

for the data level: a description of the heterogeneous data pool that includes a list of the available data sets and the relationships between them (equivalence and part/whole relations).

for the view level: a description of the visual representations consisting of a list of the available views and a statement of their applicability to the different data sets.

for the task level: a description of the analysis tasks including a list of the possible tasks and an assignment of the individual analysis tasks to appropriate views to perform them on visually and to appropriate data sets for a direct computational analysis.

The information condensed in these descriptions helps to better understand the underlying requirements and to externalize implicit interdependencies influencing the analysis process. These descriptions are not necessarily restricted to graphs as data, graph visualizations as views, and graph analysis as task – even though, this would be one way of concretizing this generic scheme of descriptions.

[7] http://www.caleydo.org/

Much has already been published regarding more or less formal descriptions of the above listed aspects of data [Hay06], view [Wil05], and task [BBP+00]. Yet even though these descriptions are available, they have usually been developed in their own right with different goals and use cases in mind. This leads to the situation, that they usually capture either data, views, or tasks very well, but do not provide the holistic view on all aspects as it is necessary. Therefore, the following will briefly detail the data, view, and task description in a way that allows their mutual integration, which is discussed thereafter.

Data Description

As mentioned above, this description combines a list of the data sets available in the possibly heterogeneous data pool and their interrelations. Sophisticated models of meta data exist towards that end [Hay06], which are able to include a wealth of additional information about a given data set, such as expiry dates, access restrictions, or provenance of data. While these models may certainly be applicable for a data description, their richness of descriptive features makes them somewhat unwieldy for the comparably simple purpose of listing the data sources and modeling their interrelations, for which a relatively simple graph-based notation suffices. The proposed approach for determining the interrelations builds upon a hierarchical ordering of the data involved. Such a hierarchical ordering can usually be observed or defined in any complex system, as described in [AA96] and exemplified in [ABKW06]. It makes use of the fact, that in most applications one can identify distinct levels of scale, which indicate "jumps" from one data domain to another – e.g., from persons through households to population. These distinct domains are called *application levels*. Depending on which application level a data set was gathered or measured, it is associated with this level simply by its resolution – e.g., whether it is gathered on a per-person level or on a per-household level. This implies a part/whole relation ("has-a" relation), e.g., a number of persons (parts) make up a household (whole). This relation can be partial, due to the fact, that not all parts of the whole may be available in a data set. In a part/whole relation, the applicability of Shneiderman's mantra is quite obvious in the data: the whole forms the overview and the individual parts are the details. But also other relations between data exhibit a similar correspondence to the mantra, e.g., specialization/generalization (the so-called "is-a" relation). Additionally, data on the same application level is often equivalent in the sense that it contains data on the same object, just acquired using different methods – e.g., personal income can be roughly inferred from the amount of income tax paid or from the amount of money spent on shopping. Other relations are certainly possible, as there may be additional, application dependent semantic connections between data sets.

View Description

The description of the views follows along the same lines as the data description does: first, a list of available overviews, filter and detail views (in a concrete visualization tool) is compiled. Then, their applicability to the different data sets listed in the data description is determined. This is basically done by considering firstly data type, as a graph visualization technique is surely not suited for, e.g., text or image data. And secondly data size

is considered, as not all visualization techniques scale up to huge data sets. Hereby, it is important to note two points:

1. A data set can have more than one view associated with it, if there are multiple fitting visualizations for it, for instance an overview and a detail view depending on the application level.

2. A view can be associated with more than one data set, if it incorporates multiple data sets, as it is often the case for compound visualizations which consist of multiple views.

The second point is different from a view being suitable for a number of different data sets. For this case, multiple instances of that view are introduced to disambiguate from the above point. An example for this are parallel coordinates which can be used for per-person and per-household data likewise, but they do not integrate both into one view.

Task Description

This final description adds the missing task knowledge to the two descriptions discussed above. Generally, it follows the operand/operator metaphor with the analysis tasks as operators and suitable views/data sets to perform them on as operands. In the spirit of visual analytics, these operators can be visual or computational nature. These tasks are then listed and described in their applicability to certain views and data sets, in the very same manner as the view description was related to the data description. This analogy even holds for the mentioned disambiguation of multiple dependencies: for example, a task may need several visual representations combined in one view at once to be successfully completed, but it is also possible for a task to be applicable to more than one visualization. Here again, an instantiation is proposed for the latter case in order to differentiate between both possibilities. For every task, preconditions can additionally specify certain requirements to be met, e.g., computational tasks like a hierarchical clustering or principal component analysis to be performed in a “Analyze First” step before visually exploring the results of the processed data. Likewise, postconditions imposing certain requirements on a task’s result, e.g., with regard to accuracy, can be formulated.

Combined Description

Once all three levels have been described, they can be automatically combined and pruned to support a specific workflow or scenario. To that end, it follows three steps: the first step combines data and view description, the second step brings in the task knowledge by combining the task description with the already combined data/view description, and the last step prunes the resulting description to leave only those parts relevant to a given analysis workflow. The goal is to yield the aforementioned seamlessness workflow that ensures a smooth transition from one data set to another in a heterogeneous data pool. But also other constraints may be desirable to optimize, e.g., minimizing the number of data sets to be analyzed. This would increase the overall efficiency of the analysis, as the computational effort of data cleaning and preprocessing is thereby reduced.

1. Combining Data and View Descriptions. The data and view descriptions are provided by the domain expert. Unless there are changes to these resources, these descriptions need to be determined only once, as they are independent of the actual analysis workflow to be performed. Their combination has implicitly already been done: the view description contains references to the data sets from the data description. Hence, the data and view descriptions can be merged using these references by connecting data sets and views. Having this combined description allows to transpose the dependencies between the data sets to the views by simply transferring part/whole and equivalence relations between two data sets to the views associated with them. This is, where the correspondence of relations between the data to Shneiderman's mantra is exploited: part/whole relations on the data level are now expressed in terms of overview and detail on the view level.

2. Combining Data/View and Task Descriptions. The task description is also provided by the domain expert and its combination with the view description follows the same procedure as outlined in the first step for data and view description: the task description contains references to the views and data sets, which can be exploited for merging both. The hierarchical dependencies, having been transferred from the data onto the view description, can now also be forwarded onto the tasks. All descriptions taken together form thus the combined description. The transferred data dependencies now running between the tasks form a graph that models all possible seamless interaction paths through the available data.

3. Pruning according to Analysis Relevance. Because of combinatorial explosion when transferring the dependencies from data to view and task level, the resulting combined description forms a very dense and hardly useful graph as it is. It is now, that the analysis workflow at hand comes into play. The workflow is defined by the analyst in the form of a sequence of tasks from the task description. This workflow can be supported seamlessly by the setup of the given scenario, if and only if a path exists in the setup's combined description that connects the tasks in the given sequential order. If such a seamless path does not exists, it is also possible to search for a path in the combined description that has the least number of gaps in it, and is thus as seamless as possible. As the combined description contains a distinct mapping between tasks, views, and data, the workflow can then be expressed in terms of views (for visual analysis tasks) and data (for computational analysis tasks), effectively realizing the mentioned *backwards direction* of task-driven design. This allows determining and pruning all views and all data sets irrelevant to the analysis workflow at hand. Additional constraints (e.g., privacy issues, when a data set is available, but may not be retrieved as the security clearance is not met) can be incorporated in this last step by pruning the nodes and edges from the combined description which should not be included in any of the possible analysis paths through the data. If multiple possible seamless paths are found, those that are excessively long and therefore inefficient, can also be disregarded.

This summarizes the fundamental procedure of constructing such a comprehensive model of data, view, and task level. How this is done for a concrete example and how it is subsequently used to guide the analysis is detailed next.

6.3.2 Constructing and Utilizing the Comprehensive Model

The scenario used here is taken from the field of life sciences and is targeted at the analysis of patient data. It stems from the ongoing research project Caleydo at the TU Graz. The analysis goal is finding a suitable treatment plan for a newly diagnosed cancer patient. This is an example of a truly complex explorative analysis workflow exhibiting all the challenges of graph analysis in the from of pathway analysis, but also including many more data sets, such as CT-images, gene expressions, etc. and a variety of visualizations to show them. The workflow of finding a treatment plan spans multiple of these data sets and views and is much more complex than a pure drill-down analysis.

Modeling this scenario follows the steps outlined in Chapter 6.3.1: describing data, views, as well as tasks, and combining these descriptions into a comprehensive model, which is then pruned according to the given workflow. To begin with, the domain expert compiles a list of all available data sets from the application domain. The underlying data of this particular use case forms a natural application hierarchy [ABKW06], which is divided into application levels, as shown in Figure 6.9.

Going down the hierarchy can be seen as an abstract zoom into the patient. When creating the **data description**, one can make use of this hierarchy, as most of the levels are in a part/whole relation. The patient level on the top contains patient information, such as age, gender, and clinical history. On the next level, the organ level, data is usually acquired using imaging techniques such as MR, CT, and X-Ray. This is one of the aforementioned cases, where multiple data has been gathered, regarding the same matter – namely the location of the tumor. Hence, there is an equivalence relation between them. Further down on tissue level, there are microscopy images of tissue samples. Below the tissue, on cell level, pathway graphs describe biochemical processes within the cell. Pathways, available in online databases, are patient independent. There are many specific pathways for different types of cancer. The nodes of the pathway graphs typically are genes and proteins. An important property of genes and proteins is their expression regulation, which is unique for an experiment. The expression regulation tells a life scientist how active a gene is, which influences the processes described in the pathways and in turn the disease itself. The expression regulation is measured for different patients, forming the data set

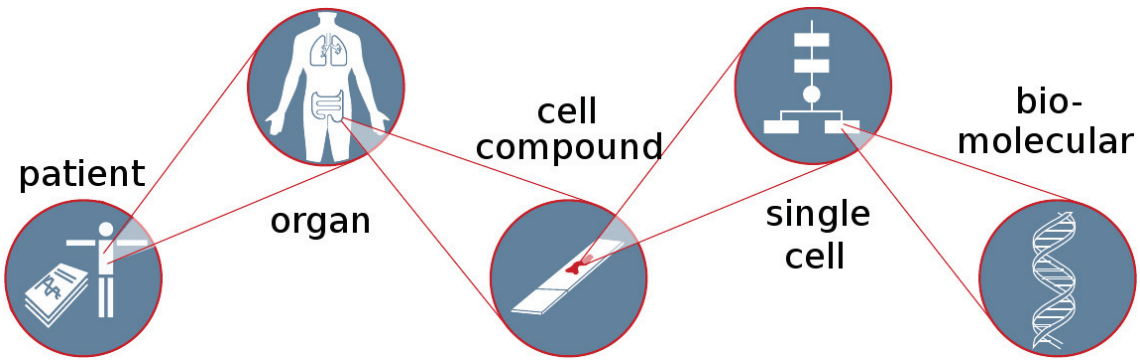


Figure 6.9: Patient-centered application hierarchy divided into application levels.

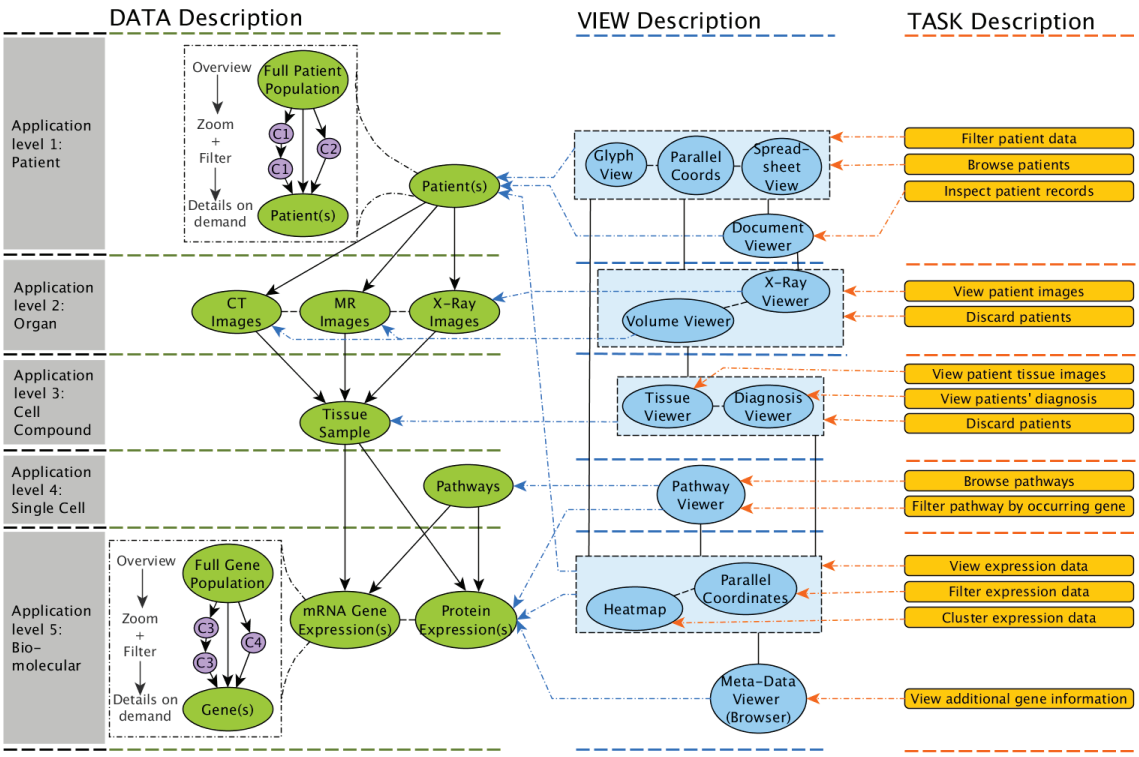


Figure 6.10: Combined description of the biomedical use case. The application levels are the semantic basis of the descriptive model in the data domain. Each data set consists of a population of data items which could be filtered down to an individual data item. Transitions between data sets describe relations among the data (part/whole = arrow, equivalence = dashed line). The blue arrows, connecting views with data sets, represent the mapping of data onto the views while the orange lines encode the associations of tasks to views. Edges between views indicate that a seamless transition between them is possible.

on the biomolecular level. Note that data descriptions are not static. Additional levels can be introduced by computational methods like hierarchical clustering of a data set. The data description is shown in Figure 6.10 in green, with the additional cluster hierarchies schematically depicted in purple.

The **view description** is composed by assigning visualizations to the data sets on their respective application level. Here, standard as well as domain specific visualization techniques are listed, as they are available in the used Caleydo software framework. It is legal to assign the same visualization technique to several levels. For instance, the parallel coordinates are useful to filter large amounts of patient data, but are also well suited as a filtering tool for the gene/protein expression data. Analogous to data sets, views can also have equivalence relations. An example would be parallel coordinates which are equal to a spreadsheet view in some aspects. They indicate potential alternative interaction paths. The view description is shown in Figure 6.10 in blue.

The complete data/view description is finally supplemented by the **task description**. Tasks are associated either with views directly or with equivalence classes of views. Tasks

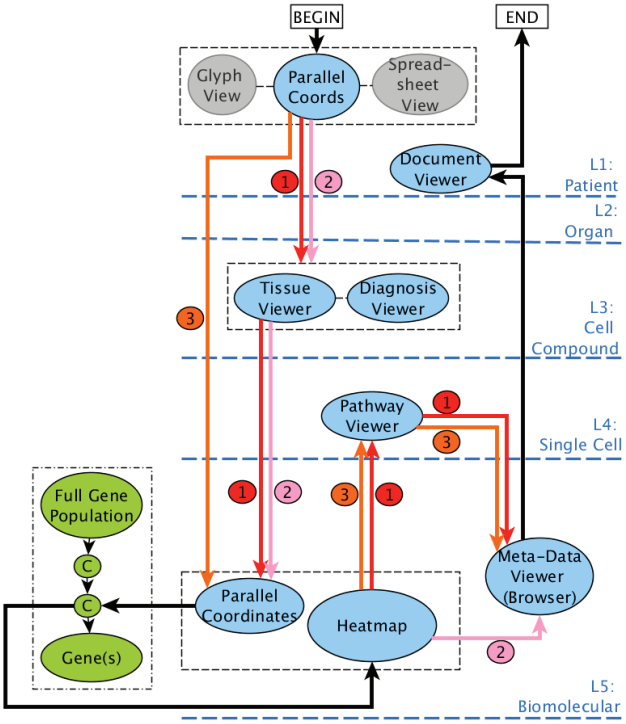


Figure 6.11: Three alternative paths for planning a treatment.

to be carried out directly on data sets are not part of this scenario, but certainly possible. To make sense to the end user who finally decides for or against a certain task in the course of an analysis, the tasks are very closely targeted towards the actual application domain and not generic like "overview" or "relate". Examples for the biomedical use case are the "Inspect patient records"-task, which is associated with the document viewer, or the "Filter patient data"-task associated with the equivalence class glyph/spreadsheet/parallel coordinates. The task description is shown in Figure 6.10 in orange.

Of the actual **combination of all three descriptions**, only the first step has been carried out in Figure 6.10 – the relationships between the data sets have been transferred to the view description along the lines of the dependencies. The second step of transferring them from the view description to the tasks is not shown due to combinatorial explosion, because of the many equivalencies defined between views in this case. The result of the third step of pruning is shown in Figure 6.11, which leaves three alternative paths to carry out the treatment planning. To that end, the workflow is formulated as one or multiple task sequences on top of the task description. Based on the mapping of tasks to views, a set of paths through the data/view description can be automatically deducted for every workflow defined. The possible paths differ in terms of their *seamlessness* (continuity across application levels) and *efficiency* (required number of analysis steps). These aspects often balance each other, as a seamless analysis path achieves its continuity through additional intermediate steps, whereas shorter paths often sacrifice continuity through jumps and short cuts across multiple application levels. An example is shown in Figure 6.11, where analysis path 3 jumps directly from the patient level to the biomolecular level, whereas

paths 1 and 2 include an intermediate analysis step on the organ level.

It is exactly this externalization of concrete, domain-specific knowledge about the analysis process, that makes the added benefit of this comprehensive model. Examples for the various possibilities arising from the knowledge gained by such a model are:

- **guidance** of the user along a path that suits given constraints best (available time for the analysis, level of knowledge of the user, etc.) and circumvents foreseeable problems (screen space or resolution not high enough for a view, no access to certain data sets, etc.)
- **speed-up** of the analysis process by prefetching data sets, and precomputing analysis results as well as layouts for visualizations that have a high likelihood of being used in the course of the analysis
- **assignment** of individual analysis steps to appropriate domain experts for the current application level in a collaborative analysis scenario (e.g., the CT-scan should be analyzed by an oncologist, whereas the gene expression profiles should be explored by a geneticist or pathologist)

Note that defining concrete analysis paths and aligning the actual analysis with them, as described above, does not forbid the analyst to diverge from the preconceived path at any time. After all, the analysis may still be an explorative one and thus hard to predefine under all possible circumstances. This is also illustrated by the integration of this concept into the biomedical visualization framework Caleydo [SLK+09, LSKS10], which is shown in Figure 6.12. In this example, the analysis path is explicitly shown at the bottom of the software, so that the analysts know at each time on which application level and at which step of the analysis process they are. All visualizations that have already been explored during the analysis are stacked at the left side of the screen, all visualizations that the users still have to explore for the treatment planning are lined up at the right side of the screen. The users are **guided through the visualizations** from left to right in the order that is predefined in the workflow. It can be seen, that the data has obviously already been fetched and visualized even for the views on the right side that are yet to be investigated, effectively realizing the mentioned **prefetching and prelayouting**. The visualization currently explored is shown in the center of the screen, with visual links running to the previous and the next visualization on both sides. These links help to put the currently investigated data item into the broader context of the last visualization and relates it to the corresponding details to be investigated in the next step. The user can step back and forth through this stack of visualizations by simply clicking directly on the desired view or data icon in the workflow schema at the bottom. A co-located, collaborative treatment planning together with the aforementioned **assignment** is conceived for the Caleydoplex setup [WLSS09], which brings Caleydo into a multi-display environment as shown in Figure 6.13.

In sum, even though the construction of the comprehensive model is costly and elaborate, the many possibilities to exploit it justify this effort. This is especially true for application domains like medicine, where a misguided analysis may result in a grave error in treatment. Hence the inclusion of domain expert knowledge is highly valuable.

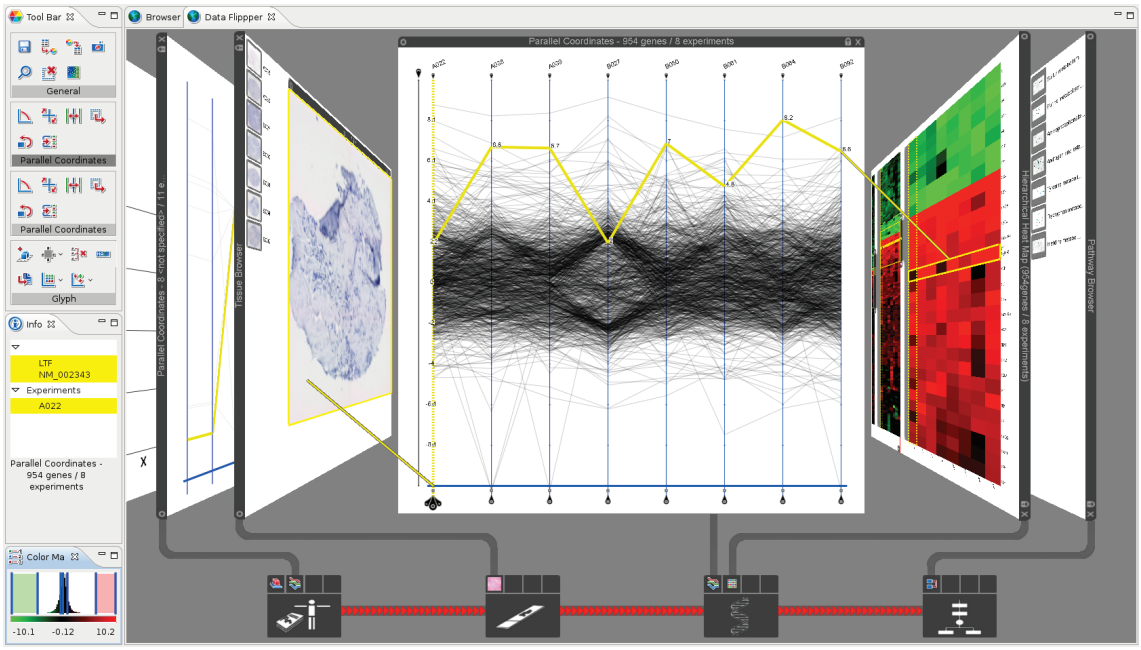


Figure 6.12: The Caleydo framework showing a parallel coordinates (PC) view in the center, a tissue viewer on the left, a heat map on the right, and additional stacked views behind them. The large symbols at the bottom represent the application levels, the small symbols on top the associated views. The red pipes depict the preferred analysis path, which, in this case, is seamless through all application levels. Visual links emphasize relations between the views (tissue slide = PC axis, PC polyline = heat map row).

6.4 Summary

This chapter picked up on the last of the three levels remaining to be discussed, the task level. It made the point, that visual exploration is:

- usually not entirely visual, as computational analysis steps are used in conjunction with the visualization, and presented a framework that integrates both aspects along the lines of Keim's Visual Analytics Mantra.
- often not completely explorative, but needs confirmative analysis steps as well – at best combined in a single representation that lets the analyst uncover hidden patterns and judge their relevance at the same time.
- not necessarily designed with the data in mind and analysis tasks to determine, but can be also perceived the other way around in case of multiple data sets, which is beneficial for devising seamless analysis.

Especially the application of the latter point to the area of graph analysis seems worthwhile to investigate. In the course of this chapter, the concept of a comprehensive, descriptive model was introduced as a general concept, but not yet spelled out concretely

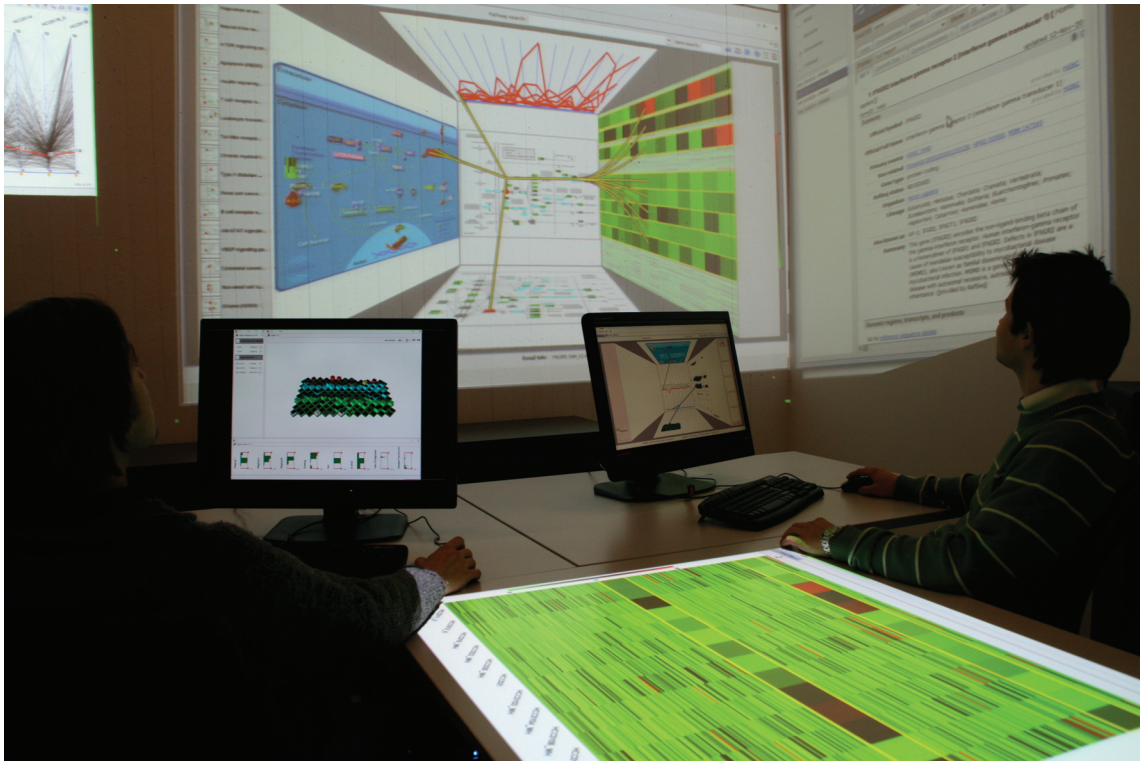


Figure 6.13: The Caleydoplex project at TU Graz aims at bringing Caleydo to a multi-display environment for co-located, collaborative visual analysis of biomedical data [WLSS09].

for the case of exploring graphs. Its applicability for this case is evident, when noting that each derived graph measure or analysis result can be perceived as a different data set, similar to the clustering results in the given example. These results from computational analysis can be placed on a hierarchy of different levels of granularity (full graph, fragments/sub graphs, neighborhoods, individual nodes/edges). Relating data from these levels would allow to move seamlessly between the different generated data sets and thus support visual graph exploration exactly on the necessary level of granularity, without the user even noticing that data set boundaries were crossed. Hence, this is a promising direction for future research.

After having discussed the implications of data, representation, and task level for explorative graph visualization, the following chapter concludes this thesis and details some follow-up questions for future work.

Chapter 7

Conclusion and Open Questions for Future Research

It was remarked in the introduction of this thesis, that "visualization as a means for exploratory graph analysis can be as intricate as it can be beneficial". In order to successfully employ visual graph representations for that purpose, it is necessary to make these intricacies explicit by naming and relating them, as it was done in Chapter 3. As a result, the three interlinked levels of data, view, and task emerged as the main influencing factors behind a useful and practical graph visualization technique supporting exploration:

- Without knowing what the **data** are (graph class and size) **expressivity** cannot be achieved.
- Without knowing the constraints of the **representation** (available screen space and layout time) **efficiency** cannot be achieved.
- Without knowing the **tasks** to be performed and in which order, **effectiveness** cannot be achieved.

The deeper the understanding of these three levels is, the more can be gained from graph visualizations that have been derived under consideration of this knowledge. As a first step towards concrete solutions, the visualization user must be made aware of the pitfalls representations pose, and the visualization designer must be supported in circumnavigating them. An example for both was given in Chapter 4 for the subset of implicit tree visualizations. It defined a parametrizable design space containing all possible representations of this kind. This concrete design space definition allowed to discuss otherwise hard to formulate representational challenges in terms of parameter combinations (e.g., compatibility of different parameter choices) and to implement it as a tool to derive suitable implicit visualizations for individual scenarios via rapid visualization prototyping.

For going beyond prototypes, it turned out that the discussion of representational issues alone is hardly sufficient. Hence, Chapter 5 took additionally the data level into account and inferred visualizations specifically for them – not mere representational prototypes, but full visualization techniques including a number of interaction techniques and allowing different extensions. As the multitude of possible input graphs cannot be discussed in

its entirety, this was done for two example scenarios: for large trees the Point-based Tree Layout was developed, and for bipartite and hypergraphs the Table-based Visualization was devised.

As explorative analysis is rarely a single-step procedure that can be fully supported with a single explorative visualization technique, and often not even with a single data set, a discussion of the task level leads to a sequence of exploration tasks – an exploration workflow. For such a workflow to be effective to its fullest, it does not only span several data sources and visualizations, but also tie in to computational graph analysis and confirmatory analysis. A general workflow that captures all these notions is the Visual Analytics Mantra, which was spelled out and prototypically implemented specifically for graphs as input data in Chapter 6. An example of the benefits of combining computational and confirmatory analysis steps with visual and exploratory analysis steps was given for a use case from social network analysis. Together with a description of the data and view level, the workflow can be used to determine suitable data sets for a given exploration task or to guide analysts through the wealth of available data and visualizations.

First steps towards extensions to the work presented in this thesis have already been made. One of them is the explorative visualization of time-varying and geospatial graph structures. These pose special challenges to graph visualization as the geolocations of the nodes fix the layout on one hand, making it hard to adapt for a more expressive or effective visualization. Yet on the other hand this adaptability is desperately needed to convey the additional variability introduced by the time-dependent nature of the graph. Exemplary solutions are currently developed for time-varying hierarchies that are embedded in a spatial context, as these have only marginally been discussed in the literature so far. Here, the general approach is based on the idea of the irregularly shaped Point-based Tree Layout from Chapter 5.1.3[1].

Another direction that is actively pursued is taking the next step after exploration and confirmation to the presentation of the results from graph analysis. So far, this has been explored for the scenario of depicting found functional dependencies between different parts of electrical systems[2]. In this case, circuit schematics and their inherent (in an analysis step predetermined) modularity are to be presented to service personnel on the small screens of mobile handheld devices. The limited screen space makes it necessary to focus on the important aspects of the circuit schematic by using two layout adaptations:

- **abstracting unimportant parts** of the circuit by combining several low-level parts into high-level "compound parts" that represent the merged elements as a functional building block. This is common practice in electrical engineering where functional groups of parts are integrated into a single IC or circuit board to ease the repair and interchangeability with updated, newer hardware.

[1] supervised diploma thesis "Visualisierung von hierarchischen Strukturen auf einer Karte mit dem Point-Based Layout" by Steffen Hadlak 2009, to appear in part as "Visualization of Hierarchies in Space and Time" at the GeoVA(t) Workshop on Geospatial Visual Analytics: Focus on Time 2010, with an extended version invited for the International Journal of Geographical Information Science

[2] appeared as "Presenting Technical Drawings on Mobile Handhelds" in Proceedings of the 18th IRMA International Conference 2007

- **folding unimportant connections** at articulation points, which are basically bottlenecks – nodes that if eliminated from the graph will result in the graph splitting up into multiple connected components. Folded connections can be unfolded by the user if their details are needed.

Using both approaches in conjunction does not only allow to reduce the information content of large circuit schematics to a displayable amount, but also to guide the user's attention towards these important parts. Here, the importance of a building block is still predefined for each possible maintenance scenario by an application expert. Yet, for the general case of displaying analysis results, importance can be derived from significance values computed during the confirmatory analysis. Overall, the approach of reducing the level of detail for contextual information can be used for driving a user or viewer towards the interesting parts of the data. Such a guidance is a valuable feature especially for the visual presentation of very large graphs, as it "blurs out" the majority of uninteresting data by folding or abstracting, and helps to perceive the important. Yet, the abstracted data is not hidden and can be unfolded interactively at any time if needed. An animated form of this concept is the progressive information visualization [RH09], which displays the important parts first and fills in the context around the important parts in later layout steps. A first prototypical implementation that combines this concept with the point-based tree visualization technique from Chapter 5.1 is currently developed.

Apart from these two extensions, the transformation of the prototypical framework for visual analytics on graphs, as it was presented in Chapter 6.1, into a fully functional software to be used as a research tool in the graduate school "dIEM oSiRiS" raises a number of additional research questions:

On data level, the question of how to efficiently store a graph in relational databases is still open for general graphs. It can be seen from the example given in Chapter 6.2, that the way of storing the graph is essential for a smooth visual and computational analysis of large graphs. Yet, which storage method is best suited to support which visual representation and which computational analysis task, is still subject to research.

On representation level, it seems worthwhile to explore a combination of parametrizing a visualization and different measures for layout quality as they were used in Chapter 5.1. These can be perceived as descriptive measures not of the input to the analytic kernel (the graph), but of its output (the visualization). Given such a combination, is it possible to attempt an optimization of the visualization parameters, e.g., to minimize overplotting% and to maximize the Ink-Paper-ratio?

On task level, it remains to translate the comprehensive model of interactive analysis from Chapter 6.3 to the case of graphs and then integrate it with the software framework. Another interesting question raised by this is how to generalize the comprehensive workflow model for domains that do not exhibit a hierarchical ordering?

Finding answers to these questions will be the first stepping stones towards a full understanding of the processes and influencing factors of a comprehensive visual graph analysis – and a feature-rich software framework to go along with it.

Bibliography

[AA96] Valerie Ahl and T.F.H. Allen. *Hierarchy Theory: A Vision, Vocabulary, and Epistomology*. Columbia University Press, 1996. ISBN 0231084811.

[ABB+00] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000. doi:10.1038/75556.

[ABKW06] Ernesto Andrianantoandro, Subhayu Basu, David K. Karig, and Ron Weiss. Synthetic biology: new engineering rules for an emerging discipline. *Molecular Systems Biology*, 2(2006.0028), 2006. doi:10.1038/msb4100073.

[ADD+04] David Auber, Maylis Delest, Jean-Philippe Domenger, Philippe Duchon, and Jean-Marc Fédou. New Strahler numbers for rooted plane trees. In Michael Drmota, Philippe Flajolet, Danièle Gardy, and Bernhard Gittenberger, editors, *Proceedings of the Colloquium on Mathematics and Computer Science Algorithms*, pages 203–215. Birkhäuser, 2004. ISBN 3764371285.

[ADH98] Armen S. Asratian, Tristan M. J. Denley, and Roland Häggkvist. *Bipartite Graphs and their Applications*. Cambridge University Press, 1998. ISBN 052159345X.

[AES05] Robert Amar, James Eagan, and John Stasko. Low-level components of analytic activity in information visualization. In John Stasko and Matt Ward, editors, *InfoVis'05: Proceedings of the IEEE Symposium on Information Visualization*, pages 111–117. IEEE Computer Society, 2005. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532136.

[AFK01] James Abello, Irene Finocchi, and Jeffrey Korn. Graph sketches. In Keith Andrews, Steven Roth, and Pak Chung Wong, editors, *InfoVis'01: Proceedings of the IEEE Symposium on Information Visualization*, pages 67–70. IEEE Computer Society, 2001. ISBN 0769513425. doi:10.1109/INFVIS.2001.963282.

[AH98] Keith Andrews and Helmut Heidegger. Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs. In Graham Wills and John Dill, editors, *InfoVis'98: Proceedings of the IEEE Symposium on Information Visualization*, pages 9–12. IEEE Computer Society, 1998. ISBN 0818690933. Late Breaking Hot Topic Paper.

[AK07] Keith Andrews and Janka Kasanicka. A comparative study of four hierarchy browsers using the hierarchical visualisation testing environment (HVTE). In *IV'07: Proceedings of the International Conference on Information Visualisation*, pages 81–86. IEEE Computer Society, 2007. ISBN 0769529003. doi:10.1109/IV.2007.8.

[AKS+02] Keith Andrews, Wolfgang Kienreich, Vedran Sabol, Jutta Becker, Georg Droschl, Frank Kappe, Michael Granitzer, Peter Auer, and Klaus Tochtermann. The InfoSky visual explorer: Exploiting hierarchical structure and document similarities. *Information Visualization*, 1(3/4):166–181, 2002. doi:10.1057/palgrave.ivs.9500023.

[AMA07] Daniel Archambault, Tamara Munzner, and David Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, March 2007. doi: 10.1109/TVCG.2007.46.

[Ank01] Mihael Ankerst. *Visual Data Mining*. dissertation.de, 2001. ISBN 3898252019.

[AOPW01] Steven R. Abrams, Daniel V. Oppenheim, Donald P. Pazel, and James L. Wright. Method and apparatus for displaying and navigating a graph, United States Patent 6285367 B1, filed May 26, 1998, published Sept. 4, 2001.

[AS04] Robert A. Amar and John T. Stasko. A knowledge task-based framework for design and evaluation of information visualizations. In Matt Ward and Tamara Munzner, editors, *InfoVis'04: Proceedings of the IEEE Symposium on Information Visualization*, pages 143–149. IEEE Computer Society, 2004. ISBN 0780387791. doi:10.1109/INFOVIS.2004.10.

[AvH04] James Abello and Frank van Ham. Matrix zoom: A visual interface to semi-external graphs. In Matt Ward and Tamara Munzner, editors, *InfoVis'04: Proceedings of the IEEE Symposium on Information Visualization*, pages 183–190. IEEE Computer Society, 2004. ISBN 0780387791. doi: 10.1109/INFVIS.2004.46.

[AWP97] Keith Andrews, Josef Wolte, and Michael Pichler. Information pyramids™: A new approach to visualising large hierarchies. In Roni Yagel and Hans Hagen, editors, *Visualization'97: Proceedings of the IEEE Conference on Visualization*, pages 49–52. ACM Press, 1997. ISBN 1581130112. Late Breaking Hot Topic Paper.

[Bar69] Margaret E. Baron. A note on the historical development of logic diagrams: Leibniz, Euler and Venn. *The Mathematical Gazette*, 53(284):113–125, 1969.

[BBP+00] Todd Bentley, Sandrine Balbo, Cécile Paris, Jean-Claude Tarby, and Lorraine Johnston. Task modelling review of methods, tools, and other. Technical Report CSIRO-MIS 2000/155, Commonwealth Scientific and Industrial Research Organisation, Sydney, Australia, August 2000.

[BCL+07] Romain Bourqui, Ludovic Cottret, Vincent Lacroix, David Auber, Patrick Mary, Marie-France Sagot, and Fabien Jourdan. Metabolic network visualization eliminating node redundance and preserving metabolic pathways. *BMC Systems Biology*, 1(29), 2007. doi:10.1186/1752-0509-1-29.

[BCS04] Thomas Bladh, David A. Carr, and Jeremiah Scholl. Extending tree-maps to three dimensions: A comparative study. In Masood Masoodian, Steve Jones, and Bill Rogers, editors, *APHI'04: Proceedings of the Asia Pacific Conference on Computer Human Interaction*, Lecture Notes in Computer Science, pages 50–59. Springer, 2004. ISBN 9783540223122. doi:10.1007/b98382.

[BD04] Michael Balzer and Oliver Deussen. Hierarchy based 3D visualization of large software structures. In *Visualization'04: Poster Compendium of the IEEE Conference on Visualization*, pages 81–82. 2004. doi:10.1109/VISUAL.2004.39.

[BD05] Michael Balzer and Oliver Deussen. Voronoi treemaps. In John Stasko and Matt Ward, editors, *InfoVis'05: Proceedings of the IEEE Symposium on Information Visualization*, pages 49–56. IEEE Computer Society, 2005. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532128.

[BE00] François Bertault and Peter Eades. Drawing hypergraphs in the subset standard. In Joe Marks, editor, *GD'00: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 45–76. Springer, 2000. ISBN 9783540415541. doi:10.1007/3-540-44541-2_15.

[Bed01] Benjamin B. Bederson. PhotoMesa: A zoomable image browser using quantum treemaps and bubblemaps. In Joe Marks and Elizabeth D. Mynatt, editors, *UIST'01: Proceedings of the Annual ACM Symposium on User Interface Software and Technology*, pages 71–80. ACM Press, 2001. ISBN 158113438X. doi:10.1145/502348.502359.

[BEF+98] Prosenjit Bose, Hazel Everett, Sándor P. Fekete, Michael E. Houle, Anna Lubiw, Henk Meijer, Kathleen Romanik, Günter Rote, Thomas C. Shermer, Sue Whitesides, and Christian Zelle. A visibility representation for graphs in three dimensions. *Journal of Graph Algorithms and Applications*, 2(3):1–16, 1998.

[Beh97] John T. Behrens. Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131–160, June 1997. doi:10.1037/1082-989X.2.2.131.

[Ber81] Jacques Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter, 1981. ISBN 3110088681.

[Ber89] Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*. Elsevier Science Publishers, 1989. ISBN 0444874895.

[BETT99] Guiseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999. ISBN 0133016153.

[BG91] Jean-Pierre Barthélemy and Alain Guénoche. *Trees and Proximity Representations*. John Wiley and Sons, 1991. ISBN 0471922633.

[BHvW00] Mark Bruls, Kees Huizing, and Jarke van Wijk. Squarified treemaps. In Wim de Leeuw and Robert van Liere, editors, *Data Visualization 2000: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 33–42. Springer, 2000. ISBN 3211835156.

[BJL02] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Improving Walker's algorithm to run in linear time. In Michael T. Goodrich and Stephen G. Kobourov, editors, *GD'02: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 347–364. Springer, 2002. ISBN 9783540001584. doi:10.1007/3-540-36151-0_32.

[Bla06] Thomas Bladh. *Towards an Understanding of Dynamics in Information Visualization*. Ph.D. thesis, LuleåUniversity of Technology, June 2006.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: A survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999. ISBN 089871432X.

[BMS97] Prosenjit Bose, Michael McAllister, and Jack Snoeyink. Optimal algorithms to embed trees in a point set. *Journal of Graph Algorithms and Applications*, 1(2):1–15, 1997.

[BN01] Todd Barlow and Padraic Neville. A comparison of 2-D visualizations of hierarchies. In Keith Andrews, Steven Roth, and Pak Chung Wong, editors, *InfoVis'01: Proceedings of the IEEE Symposium on Information Visualization*, pages 131–138. IEEE Computer Society, 2001. ISBN 0769513425. doi:10.1109/INFVIS.2001.963290.

[BPV96] Luc Beaudoin, Marc-Antoine Parent, and Louis C. Vroomen. Cheops: A compact explorer for complex hierarchies. In Roni Yagel and Gregory M. Nielson, editors, *Visualization'96: Proceedings of the IEEE Conference on Visualization*, pages 87–92. ACM Press, 1996. ISBN 0897918649.

[BR05] Danail Bonchev and Dennis H Rouvray, editors. *Complexity in Chemistry, Biology, and Ecology*. Springer, 2005. ISBN 9780387232645. doi:10.1007/b136300.

[Bra01] Ulrik Brandes. Drawing on physical analogies. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, Lecture Notes in Computer Science, pages 71–86. Springer, 2001. ISBN 9783540420620. doi:10.1007/3-540-44969-8_4.

[BSW02] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, 2002. doi:10.1145/571647.571649.

[Bug05] Malgorzata Bugajska. Framework for spatial visual design of abstract information. In Ebad Banissi, Muhammad Sarfraz, Jonathan C. Roberts, Bowen Loften, Anna Ursyn, Remo Aslak Burkhard, Angela Lee, and Gennady Andrienko, editors, *IV'05: Proceedings of the International Conference Information Visualisation*, pages 713–723. IEEE Computer Society, 2005. ISBN 0769523978. doi:10.1109/IV.2005.51.

[BW98] Ulrik Brandes and Dorothea Wagner. Dynamic grid embedding with few bends and changes. In Oscar H. Ibarra and Kyung-Yong Chwa, editors, *ISAAC'98: Proceedings of the International Symposium on Algorithms and Computations*, Lecture Notes in Computer Science, pages 89–99. Springer, 1998. ISBN 9783540653851. doi:10.1007/3-540-49381-6_11.

[CAT07] Fanny Chevalier, David Auber, and Alexandru Telea. Structural analysis and visualization of C++ code evolution using syntax trees. In Massimiliano Di Penta and Michele Lanza, editors, *IWPSE'07: Proceedings of the International workshop on Principles of Software Evolution*, pages 90–97. ACM Press, 2007. ISBN 9781595937223. doi:10.1145/1294948.1294971.

[CBDL06] Evelyn Camon, Daniel Barrell, Emily Dimmer, and Vivian Lee. The gene ontology annotation (GOA) database. In Nicola Mulder and Rolf Apweiler, editors, *In Silico Genomics And Proteomics: Functional Annotation of Genomes And Proteins*, chapter 4, pages 37–54. Nova Science Publishers, 2006. ISBN 1594549958.

[Cel04] Joe Celko. *Trees and Hierarchies in SQL for Smarties*. Morgan Kaufmann Publishers, 2004. ISBN 1558609202.

[Che05] Chaomei Chen. Top 10 unsolved information visualization problems. *IEEE Computer Graphics and Applications*, 25(4):12–16, 2005. doi: 10.1109/MCG.2005.91.

[Che06] Chaomei Chen. *Information Visualization. Beyond the Horizon*. Springer, 2nd edition, 2006. ISBN 184628340X. doi:10.1007/1-84628-579-8.

[Chu98] Mei C. Chuah. Dynamic aggregation with circular visual designs. In Graham Wills and John Dill, editors, *InfoVis'98: Proceedings of the IEEE Symposium on Information Visualization*, pages 35–43. IEEE Computer Society, 1998. ISBN 0818690933. doi:10.1109/INFVIS.1998.729557.

[CHZ+07] Bas Cornelissen, Danny Holten, Andy Zaidman, Leon Moonen, Jarke J. van Wijk, and Arie van Deursen. Understanding execution traces using massive sequence and circular bundle views. In Kenny Wong, Eleni Stroulia, and Paolo Tonella, editors, *ICPC'07: Proceedings of the IEEE International Conference on Program Comprehension*, pages 49–58. IEEE Computer Society, 2007. ISBN 0769528600. doi:10.1109/ICPC.2007.39.

[Cim06] Robert Cimikowski. An analysis of some linear graph layout heuristics. *Journal of Heuristics*, 12(3):143–153, May 2006. doi:10.1007/s10732-006-4294-9.

[CKI99] Neville Churcher, Lachlan Keown, and Warwick Irwin. Virtual worlds for software visualisation. In *SoftVis'99: Proceedings of the Software Visualisation Workshop*, pages 9–16. 1999.

[CM04] Andrew Cockburn and Bruce McKenzie. Evaluating spatial memory in two and three dimensions. *International Journal of Human-Computer Studies*, 61(3):359–373, 2004. doi:10.1016/j.ijhcs.2004.01.005.

[CS96] Robert Cimikowski and Paul Shope. A neural-network algorithm for a graph layout problem. *IEEE Transactions on Neural Networks*, 7(2):341–345, March 1996. doi:10.1109/72.485670.

[CS09] Abon Chaudhuri and Han-Wei Shen. A self-adaptive treemap-based technique for visualizing hierarchical data in 3D. In Peter Eades, Thomas Ertl, and Han-Wei Shen, editors, *PacificVis'09: Proceedings of the IEEE Pacific Visualization Symposium*, pages 105–112. IEEE Computer Society, 2009. ISBN 9781424444055. doi:10.1109/PACIFICVIS.2009.4906844.

[CVW09] Christopher Collins, Fernanda B. Viégas, and Martin Wattenberg. Parallel tag clouds to explore and analyze faceted text corpora. In John Stasko and Jarke J. van Wijk, editors, *VAST'09: Proceedings of the IEEE Symposium on Visual Analytics Science and Technology*, pages 91–98. IEEE Computer Society, 2009. ISBN 9781424452835. doi:10.1109/VAST.2009.5333443.

[DBJ+07] Natalie C. Duarte, Scott A. Becker, Neema Jamshidi, Ines Thiele, Monica L. Mo, Thuy D. Vo, Rohith Srivas, and Bernhard Ø. Palsson. Global reconstruction of the human metabolic network based on genomic and bibliomic data. *Proceedings of the National Academy of Sciences of the USA*, 104(6):1777–1782, February 2007. doi:10.1073/pnas.0610772104.

[DBW00] Alan Dix, Russell Beale, and Andy Wood. Architectures to make simple visualisations using simple systems. In Vito Di Gesú, Stefano Levialdi, and Laura Tarantino, editors, *AVI'00: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 51–60. ACM Press, 2000. ISBN 1581132522. doi:10.1145/345513.345250.

[DEMHP07] Alice M. Dean, Joanna A. Ellis-Monaghan, Sarah Hamilton, and Greta Pangborn. Unit rectangle visibility graphs. *The Electronic Journal of Combinatorics*, 15(1 - #R79), 2007.

[DES09] Matthias Dehmer and Frank Emmert-Streib. Towards network complexity. In Jie Zhou, editor, *Complex'09: Proceedings of the International Conference Complex*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 707–714. Springer, 2009. ISBN 9783642024658. doi:10.1007/978-3-642-02466-5_68.

[DH94] Alice M. Dean and Joan P. Hutchinson. Rectangle-visibility representations of bipartite graphs. In Roberto Tamassia and Ioannis G. Tollis, editors, *GD'94: Proceedings of the DIMACS International Workshop on Graph Drawing*, Lecture Notes in Computer Science, pages 159–166. Springer, 1994. ISBN 9783540589501. doi:10.1007/3-540-58950-3_367.

[DH02] Helmut Doleisch and Helwig Hauser. Smooth brushing for focus+context visualization of simulation data in 3d. In *WSCG'02: Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 147–154. 2002.

[DK08] Jiří Dokulil and Jana Katreniaková. Edge routing with fixed node positions. In Ebad Banissi, Liz Stuart, Mikael Jern, Gennady Andrienko, Francis T. Marchese, Nasrullah Memon, Reda Alhajj, Theodor G. Wyeld, Remo Aslak Burkhard, Georges Grinstein, Dennis Groth, Anna Ursyn, Carsten Maple, Anthony Faiola, and Brock Craft, editors, *IV'08: Proceedings of the International Conference on Information Visualisation*, pages 626–631. IEEE Computer Society, 2008. ISBN 9780769532684. doi:10.1109/IV.2008.11.

[DKMT07] Ugur Dogrusoz, Konstantinos G. Kakoulisc, Brendan Maddena, and Ioannis G. Tollis. On labeling in graph visualization. *Information Sciences*, 177(12):2459–2472, June 2007. doi:10.1016/j.ins.2007.01.019.

[DLR09] Geoffrey M. Draper, Yarden Livnat, and Richard F. Riesenfeld. A survey of radial methods for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15:759–776, 2009. doi:10.1109/TVCG.2009.23.

[dM99] Patrice Ossona de Mendez. The reduced genus of a multigraph. In Christoph Meinel and Sophie Tison, editors, *STACS'99: Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, pages 16–31. Springer, 1999. ISBN 9783540656913. doi:10.1007/3-540-49116-3_2.

[DMG+08] Joan DiMicco, David R. Millen, Werner Geyer, Casey Dugan, Beth Brownholtz, and Michael Muller. Motivations for social networking at work. In *CSCW'08: Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 711–720. ACM Press, 2008. ISBN 9781605580074. doi:10.1145/1460563.1460674.

[DMS05] Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast node overlap removal. In Patrick Healy and Nikola S. Nikolov, editors, *GD'05: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 153–164. Springer, 2005. ISBN 9783540314257. doi:10.1007/11618058_15.

[DMS06] Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast node overlap removal – correction. In Michael Kaufmann and Dorothea Wagner, editors, *GD'06: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 446–447. Springer, 2006. ISBN 9783540709039. doi:10.1007/978-3-540-70904-6_44.

[dNMB05] Wouter de Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek*. Structural Analysis in the Social Sciences. Cambridge University Press, 2005. ISBN 0521602629.

[DPS02] Josep Díaz, Jordi Petit, and María Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002. doi:10.1145/568522.568523.

[ED06] Geoffrey Ellis and Alan Dix. The plot, the clutter, the sampling and its lens: occlusion measures for automatic clutter reduction. In Augusto Celentano and Piero Mussio, editors, *AVI'06: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 266–269. ACM Press, 2006. ISBN 1595933530. doi:10.1145/1133265.1133318.

[EDG+08] Niklas Elmqvist, Thanh-Nghi Do, Howard Goodell, Nathalie Henry, and Jean-Daniel Fekete. ZAME: Interactive large-scale graph visualization. In Issei Fujishiro, Hua Li, and Kwan-Liu Ma, editors, *PacificVis'08: Proceedings of the IEEE Pacific Visualization Symposium*, pages 215–222.

IEEE Computer Society, 2008. ISBN 9781424419661. doi:10.1109/PACIFICVIS.2008.4475479.

[EK02] Stephen G. Eick and Alan F. Karr. Visual scalability. *Journal of Computational and Graphical Statistics*, 11(1):22–43, March 2002.

[FP02] Jean-Daniel Fekete and Catherine Plaisant. Interactive information visualization of a million items. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 117–124. IEEE Computer Society, 2002. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173156.

[FP03] Jean-Daniel Fekete and Catherine Plaisant. TreeML DTD describing a tree structure for visualization. `http://www.nomencurator.org/InfoVis2003/download/treeml.dtd`, 2003. Retrieved 28-APR-2010.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, November 1991. doi:10.1002/spe.4380211102.

[Fre04] Linton C. Freeman. *The Development of Social Network Analysis: A Study in the Sociology of Science*. Empirical Press, 2004. ISBN 1594577145.

[FT94] Andrew U. Frank and Sabine Timpf. Multiple representations for cartographic objects in a multi-scale tree – an intelligent graphical zoom. *Computers and Graphics*, 18(6):823–829, November-December 1994. doi:10.1016/0097-8493(94)90008-6.

[FT07] Yaniv Frishman and Ayellet Tal. Multi-level graph layout on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1310–1319, November-December 2007. doi:10.1109/TVCG.2007.70580.

[Fur86] George W. Furnas. Generalized fisheye views. *ACM SIGCHI Bulletin*, 17(4):16–23, April 1986. doi:10.1145/22339.22342.

[FWD+03] Jean-Daniel Fekete, David Wang, Niem Dang, Aleks Aris, and Catherine Plaisant. Overlaying graph links on treemaps. In *InfoVis'03: Poster Compendium of the IEEE Symposium on Information Visualization*, pages 82–83. 2003.

[FWR99] Ying-Huey Fua, Matthew O. Ward, and Elke A. Rundensteiner. Navigating hierarchies with structure-based brushes. In Daniel Keim and Graham Wills, editors, *InfoVis'99: Proceedings of the IEEE Symposium on Information Visualization*, pages 58–64. IEEE Computer Society, 1999. ISBN 0769504310. doi:10.1109/INFVIS.1999.801858.

[GADM04] Sébastien Grivet, David Auber, Jean-Philippe Domenger, and Guy Melançon. Bubble tree drawing algorithm. In Konrad Wojciechowski, Bogdan Smolka, Henryk Palus, Ryszard Kozera, Wladyslaw Skarbek, and Lyle Noakes, editors, *ICCVG'04: Proceedings of the International Conference on Computer Vision and Graphics*, pages 633–641. Springer, 2004. ISBN 9781402041785. doi:10.1007/1-4020-4179-9_91.

[GFC05] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, July 2005. doi:10.1057/palgrave.ivs.9500092.

[GGL08] Emilio Di Giacomo, Luca Grilli, and Giuseppe Liotta. Drawing bipartite graphs on two parallel convex curves. *Journal of Graph Algorithms and Applications*, 12(1):97–112, 2008.

[GHGH08] Apeksha Godiyal, Jared Hoberock, Michael Garland, and John C. Hart. Rapid multipole graph drawing on the GPU. In Ioannis G. Tollis and Maurizio Patrignani, editors, *GD'08: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 90–101. Springer, 2008. ISBN 9783642002182. doi:10.1007/978-3-642-00219-9_10.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness*. W.H.Freeman, 1979. ISBN 0716710447.

[GK06] Emden R. Gansner and Yehuda Koren. Improved circular layouts. In Michael Kaufmann and Dorothea Wagner, editors, *GD'06: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 386–398. Springer, 2006. ISBN 9783540709039. doi:10.1007/978-3-540-70904-6_37.

[GLPN93] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, 1993. doi:10.1016/0166-218X(93)90045-P.

[GP07] Markus Gross and Hanspeter Pfister, editors. *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007. ISBN 0123706041.

[GR03] Ashim Garg and Adrian Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In Vipin Kumar, Marina L. Gavrilova, Chih Jeng, Kenneth Tan, and Pierre L'Ecuyer, editors, *ICCSA'03: Proceedings of the International Conference on Computational Science and Its Applications*, Lecture Notes in Computer Science, pages 876–885. Springer, 2003. ISBN 9783540401568. doi:10.1007/3-540-44842-X_89.

[GS98] Giorgio Gallo and Maria Grazia Scutella. Directed hypergraphs as a modelling paradigm. *Decisions in Economics and Finance*, 21(1-2):97–123, June 1998. doi:10.1007/BF02735318.

[Hay06] David C. Hay. *Data Model Patterns. A Metadata Map*. Morgan Kaufmann Publishers, 2006. ISBN 0120887983.

[Hen08] Nathalie Henry. *Exploring Social Networks with Matrix-based Representations*. Ph.D. thesis, Université Paris-Sud, France, July 2008.

[HF06] Nathalie Henry and Jean-Daniel Fekete. MatrixExplorer: A dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):677–684, September-October 2006. doi:10.1109/TVCG.2006.160.

[HF07] Nathalie Henry and Jean-Daniel Fekete. MatLink: Enhanced matrix visualization for analyzing social networks. In Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, *INTERACT'07: Proceedings of the IFIP TC13 Conference on Human-Computer Interaction, Part II*, Lecture Notes in Computer Science, pages 288–302. Springer, 2007. ISBN 9783540747994. doi: 10.1007/978-3-540-74800-7_24.

[HFM07] Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. NodeTrix: A hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, November/December 2007. doi:10.1109/TVCG.2007.70582.

[HHE05] Weidong Huang, Seok-Hee Hong, and Peter Eades. Layout effects on sociogram perception. In Patrick Healy and Nikola S. Nikolov, editors, *GD'05: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 262–273. Springer, 2005. ISBN 9783540314257. doi:10.1007/11618058_24.

[HHZ09] Mao Lin Huang, Tze-Haw Huang, and Jiawan Zhang. TreemapBar: Visualizing additional dimensions of data in bar chart. In Ebad Banissi, Liz Stuart, Theodor G. Wyeld, Mikael Jern, Gennady Andrienko, Nasrullah Memon, Reda Alhajj, Remo Aslak Burkhard, Georges Grinstein, Dennis Groth, Anna Ursyn, Jimmy Johansson, Camilla Forsell, Urska Cvek, Marjan Trutschi, Francis T. Marchese, Carsten Maple, Andrew J. Cowell, and Andrew Vande Moere, editors, *IV'09: Proceedings of the International Conference on Information Visualisation*, pages 98–103. IEEE Computer Society, 2009. ISBN 9780769537337. doi:10.1109/IV.2009.22.

[HMM+99] Ivan Herman, Scott Marshall, Guy Melançon, D. J. Duke, M. Delest, and J.-P. Domenger. Skeletal images as visual cues in graph visualization. In

Eduard Gröller, Helwig Löffelmann, and William Ribarsky, editors, *VisSym'99: Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 13–22. Springer, 1999.

[HNLH07] Mao Lin Huang, Quang Vinh Nguyen, Wei Lai, and Xiaodi Huang. Three-dimensional EncCon tree. In Ebad Banissi, Muhammad Sarfraz, and Natasha Dejdumrong, editors, *CGIV'07: Proceedings of the Computer Graphics, Imaging and Visualisation*, pages 429–433. IEEE Computer Society, 2007. ISBN 0769529283. doi:10.1109/CGIV.2007.82.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, September-October 2006. doi:10.1109/TVCG.2006.147.

[HONA03] Martin Hicks, Claire O'Malley, Sarah Nichols, and Ben Anderson. Comparison of 2D and 3D representations for visualising telecommunication usage. *Behavior and Information Technology*, 22(3):185–201, May 2003. doi:10.1080/0144929031000117080.

[HR09] Jan Himmelspach and Matthias Röhl. JAMES II – experiences and interpretations. In Adelinde M. Uhrmacher and Danny Weyns, editors, *Multi-Agent Systems: Simulation and Applications*, Computational Analysis, Synthesis, and Design of Dynamic Models Series, chapter 17, pages 509–533. CRC Press, 2009. ISBN 9781420070231.

[HVvW05] Danny Holten, Roel Vliegen, and Jarke J. van Wijk. Visual realism for the visualization of software metrics. In Andrian Marcus, Jonathan I. Maletic, Margaret-Anne Storey, Michele Lanza, and Stéphane Ducasse, editors, *VisSoft'05: Proceedings of the IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 27–32. IEEE Computer Society, 2005. ISBN 0780395409. doi:10.1109/VISSOF.2005.1684299.

[HvW09a] Danny Holten and Jarke J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009. doi:10.1111/j.1467-8659.2009.01450.x.

[HvW09b] Danny Holten and Jarke J. van Wijk. A user study on visualizing directed edges in graphs. In Dan R. Olsen, Jr., Richard B. Arthur, Ken Hinckley, Meredith Ringel Morris, Scott E. Hudson, and Saul Greenberg, editors, *CHI'09: Proceedings of the International Conference on Human Factors in Computing Systems*, pages 2299–2308. ACM Press, 2009. ISBN 9781605582467. doi:10.1145/1518701.1519054.

[HZ07] Jie Hao and Kang Zhang. A mobile interface for hierarchical information visualization and navigation. In *ISCE'07: Proceedings of the IEEE Inter-*

national Symposium on Consumer Electronics. IEEE Computer Society, 2007. ISBN 9781424411092. doi:10.1109/ISCE.2007.4382214.

[HZBK08] Harry Halpin, David J. Zielinski, Rachael Brady, and Glenda Kelly. Redgraph: Navigating semantic web networks using virtual reality. In Ming Lin, Anthony Steed, and Carolina Cruz-Neira, editors, *VR'08: Proceedings of the IEEE Virtual Reality Conference*, pages 257–258. IEEE Computer Society, 2008. ISBN 9781424419715. doi:10.1109/VR.2008.4480789.

[HZGZ09] Jie Hao, Kang Zhang, Chad Allen Gabrysch, and Qiaoming Zhu. Managing hierarchical information on small screens. In Qing Li, Ling Feng, Jian Pei, Sean X.Wang, Xiaofang Zhou, and Qiao-Ming Zhu, editors, *Advances in Data and Web Management: Proceedings of the Joint International Conferences APWeb/WAIM 2009*, pages 429–441. Springer, 2009. ISBN 9783642006715. doi:10.1007/978-3-642-00672-2_38.

[HZH07] Jie Hao, Kang Zhang, and Mao Lin Huang. RELT – visualizing trees on mobile devices. In Guoping Qiu, Clement Leung, Xiangyang Xue, and Robert Laurini, editors, *VISUAL'07: Advances in Visual Information Systems*, Lecture Notes in Computer Science, pages 344–357. Springer, 2007. ISBN 9783540764137. doi:10.1007/978-3-540-76414-4_34.

[IKIY02] Takayuki Itoh, Yasumasa Kajinaga, Yuko Ikehata, and Yumi Yamaguchi. Data jewelry box: A graphics showcase for large-scale hierarchical data visualization. Technical Report RT0427, IBM Research, 2002.

[IMT09] Takao Ito, Kazuo Misue, and Jiro Tanaka. Sphere anchored map: A visualization technique for bipartite graphs in 3D. In Julie A. Jacko, editor, *Human-Computer Interaction. Novel Interaction Methods and Techniques: Proceedings of the HCI International*, Lecture Notes in Computer Science, pages 811–820. Springer, 2009. ISBN 9783642025761. doi:10.1007/978-3-642-02577-8_89.

[IYIK04] Takayuki Itoh, Yumi Yamaguchi, Yuko Ikehata, and Yasumasa Kajinaga. Hierarchical data visualization using a fast rectangle-packing algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):302–313, May-June 2004. doi:10.1109/TVCG.2004.1272729.

[Jer99] Mikael Jern. Visual intelligence – turning data into knowledge. In Farzad Khosrowshahi, Muhammad Sarfraz, Eric W. Tatham, and Anna Ursyn, editors, *IV'99: Proceedings of the International Conference on Information Visualisation*, pages 3–8. IEEE Computer Society, 1999. ISBN 0769502105. doi:10.1109/IV.1999.781525.

[JLNU08] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Adelinde M. Uhrmacher. The attributed pi calculus. In Monika Heiner and Adelinde M.

Uhrmacher, editors, *CMSB'08: Proceedings of the International Conference Computational Methods in Systems Biology*, Lecture Notes in Bioinformatics, pages 83–102. Springer, 2008. ISBN 9783540885610. doi: 10.1007/978-3-540-88562-7_10.

[JM97] Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(1):1–25, 1997.

[Joh93] Brian Scott Johnson. *Treemaps: Visualizing hierarchical and categorical data*. Ph.D. thesis, University of Maryland, 1993. HCIL-94-04, UMI-94-25057.

[JS91] Brian Johnson and Ben Shneiderman. Tree-Maps: A space-filling approach to the visualization of hierarchical information structures. In Gregory M. Nielson and Larry Rosenblum, editors, *Visualization'91: Proceedings of the IEEE Conference on Visualization*, pages 284–291. IEEE Computer Society, 1991. ISBN 0818622458. doi:10.1109/VISUAL.1991.175815.

[JwWf03] Huang Jing-wei and Wei Wen-fang. Evolutionary graph drawing algorithms. *Wuhan University Journal of Natural Sciences*, 8(1):212–216, March 2003. doi:10.1007/BF02899481.

[KAMP73] George R. Kiss, Christine Armstrong, Robert Milroy, and James Piper. An associative thesaurus of English and its computer analysis. In Adam Jack Aitken, Richard Weld Bailey, and Neil Hamilton-Smith, editors, *The Computer and Literary Studies*. Edinburgh University Press, 1973.

[KC96] Rick Kazman and Jeromy Carriére. Rapid prototyping of information visualizations using VANISH. In Nahum D. Gershon, Stuart Card, and Stephen G. Eick, editors, *InfoVis'96: Proceedings of the IEEE Symposium on Information Visualization*, pages 21–28. IEEE Computer Society, 1996. ISBN 081867668X. doi:10.1109/INFVIS.1996.559212.

[KE00] Victor N. Kasyanov and Vladimir A. Evstigneev. *Graph Theory for Programmers - Algorithms for Processing Trees*. Mathematics and its Applications. Kluwer Academic Publishers, 2000. ISBN 0792364287.

[KEC06] René Keller, Claudia M. Eckert, and P. John Clarkson. Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? *Information Visualization*, 5:62–76, 2006. doi: 10.1057/palgrave.ivs.9500116.

[KH81] Beat Kleiner and John A. Hartigan. Representing points in many dimensions by trees and castles. *Journal of the American Statistical Association*, 76(374):260–269, June 1981.

[KHT09] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5):e1000385, May 2009. doi: 10.1371/journal.pcbi.1000385.

[KK94] Peter R. Keller and Mary M. Keller. *Visual Cues: Practical Data Visualization*. IEEE Computer Society, 1994. ISBN 0818631023.

[KL83] Joseph B. Kruskal and James M. Landwehr. Icicle plot: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[KLMS95] Harri Klemetti, Ismo Lapinleimu, Erkki Mäkinen, and Mika Sieranta. A programming project: trimming the spring algorithm for drawing hypergraphs. *ACM SIGCSE Bulletin*, 27(3):34–38, September 1995. doi: 10.1145/209849.209855.

[KMBG07] Pierre-Yves Koenig, Guy Melançon, Charles Bohan, and Berengere Gautier. Combining DagMaps and Sugiyama layout for the navigation of hierarchical data. In *IV'07: Proceedings of the International Conference on Information Visualisation*, pages 447–452. IEEE Computer Society, 2007. ISBN 0769529003. doi:10.1109/IV.2007.36.

[KMSZ06] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In Ebad Banissi, Remo Aslak Burkhard, Anna Ursyn, Jian J. Zhang, Mark Bannatyne, Carsten Maple, Andrew J. Cowell, Gui Yun Tian, and Ming Hou, editors, *IV'06: Proceedings of the International Conference on Information Visualisation*, pages 9–16. IEEE Computer Society, 2006. ISBN 0769526020. doi: 10.1109/IV.2006.31.

[Kob10] Stephen G. Kobourov. Force-directed drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*. CRC Press, 2010. To appear.

[Kre04] Matthias Kreuseler. *Ein flexibles Framework zum Visuellen Data Mining*. Ph.D. thesis, Universität Rostock, 2004.

[KS02] Matthias Kreuseler and Heidrun Schumann. A flexible approach for visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):39–51, 2002. doi:10.1109/2945.981850.

[KSB+09] Martin Krzywinski, Jacqueline Schein, Inanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 19:1639–1645, September 2009. doi:10.1101/gr.092759.109.

[KV97] Can Keskin and Volker Vogelmann. Effective visualization of hierarchical graphs with the cityscape metaphor. In David S. Ebert and Charles K. Nicholas, editors, *NPIVM'97: Proceedings of the Workshop on New*

Paradigms in Information Visualization and Manipulation, pages 52–57. ACM Press, 1997. ISBN 1581130511. doi:10.1145/275519.275531.

[KY93] Hideki Koike and Hirotaka Yoshihara. Fractal approaches for visualizing huge hierarchies. In *VL'93: Proceedings of the IEEE Workshop on Visual Languages*, pages 55–60. IEEE Computer Society, 1993. ISBN 0818639709. doi:10.1109/VL.1993.269566.

[Lam08] Heidi Lam. A framework of interaction costs in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1149—1156, November-December 2008. doi:10.1109/TVCG.2008.109.

[Lee06] Bongshin Lee. *Interactive Visualization for Trees and Graphs*. Ph.D. thesis, University of Maryland, 2006.

[LF08] Hao R. Lü and James Fogarty. Cascaded Treemaps: Examining the visibility and stability of structure in Treemaps. In Lyn Bartram and Chris Shaw, editors, *GI'08: Proceedings of the Graphics Interface Conference*, pages 259–266. Canadian Information Processing Society, 2008. ISBN 9781568814230.

[LGC09] Lin Li, Bao-Yan Gu, and Li Chen. The topological characteristics and community structure in consumer-service bipartite graph. In Jie Zhou, editor, *Complex'09: Proceedings of the International Conference Complex*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 640–650. Springer, 2009. ISBN 9783642024658. doi:10.1007/978-3-642-02466-5_62.

[LHM+09] Peter Liggesmeyer, Jens Heidrich, Jürgen Münch, Robert Kalcklösch, Henning Barthel, and Dirk Zeckzer. Visualization of software and systems as support mechanism for integrated software project control. In Julie A. Jacko, editor, *Human-Computer Interaction. Novel Interaction Methods and Techniques: Proceedings of the HCI International*, Lecture Notes in Computer Science, pages 846–855. Springer, 2009. ISBN 9783642025761. doi:10.1007/978-3-642-02574-7_94.

[LKG07] Adrian Lienhard, Adrian Kuhn, and Orla Greevy. Rapid prototyping of visualizations using Mondrian. In Jonathan I. Maletic, Alexandru Telea, and Andrian Marcus, editors, *VisSoft'07: Proceedings of the IEEE International Workshop onVisualizing Software for Understanding and Analysis*, pages 67–70. IEEE Computer Society, 2007. ISBN 1424406005. doi:10.1109/VISSOF.2007.4290702.

[LPP+06] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In Enrico Bertini, Catherine Plaisant, and Giuseppe Santucci, editors, *BELIV'06:*

Proceedings of the AVI Workshop "BEyond Time and Errors: Novel Evaluation Methods for Information Visualization", pages 1–5. ACM Press, 2006. ISBN 1595935622. doi:10.1145/1168149.1168168.

[LR96] Jonh Lamping and Ramana Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–55, March 1996. doi:10.1006/jvlc.1996.0003.

[LS08] Martin Luboschik and Heidrun Schumann. Discovering the covered: Ghost-views in information visualization. In Steve Cunningham and Vaclav Skala, editors, *WSCG'08: Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 113–118. University of West Bohemia, 2008. ISBN 9788086943152.

[LSC08] Martin Luboschik, Heidrun Schumann, and Hilko Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1237–1244, November-December 2008. doi:10.1109/TVCG.2008.152.

[LSKS10] Alexander Lex, Marc Streit, Ernst Kruijff, and Dieter Schmalstieg. Caleydo: Design and evaluation of a visual analysis framework for gene expression data in its biological context. In Stephen North, Han-Wei Shen, and Jarke van Wijk, editors, *PacificVis'10: Proceedings of the IEEE Pacific Visualization Symposium*, pages 57–64. IEEE Computer Society, 2010. ISBN 9781424466849. To appear.

[LY07] Chun-Cheng Lin and Hsu-Chun Yen. On balloon drawings of rooted trees. *Journal of Graph Algorithms and Applications*, 11(2):431–452, 2007.

[Mäk90] Erkki Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34(3-4):177–185, 1990. doi:10.1080/00207169008803875.

[MBK97] Cathleen McGrath, Jim Blythe, and David Krackhardt. The effect of spatial arrangement on judgements and errors in interpreting graphs. *Social Networks*, 19(3):223–242, August 1997. doi:10.1016/S0378-8733(96)00299-7.

[Mis06] Kazuo Misue. Drawing bipartite graphs as anchored maps. In Kazuo Misue, Kozo Sugiyama, and Jiro Tanaka, editors, *APVIS'06: Proceedings of the Asia Pacific Symposium on Information Visualization*, Conference in Research and Practice in Information Technology, pages 169–177. Australian Computer Society, 2006. ISBN 1920682414.

[MM08] Chris Muelder and Kwan-Liu Ma. Rapid graph layout using space filling curves. *IEEE Transactions on Visualization and Computer Graphics*,

14(6):1301–1308, November-December 2008. doi:10.1109/TVCG.2008.158.

[MR10] Michael J. McGuffin and Jean-Marc Robert. Quantifying the space-efficiency of 2D graphical representations of trees. *Information Visualization*, 9(2):115–140, 2010. doi:10.1057/ivs.2009.4.

[MS08] Guy Melançon and Arnaud Sallaberry. Edge metrics for visual graph analytics: A comparative study. In Ebad Banissi, Liz Stuart, Mikael Jern, Gennady Andrienko, Francis T. Marchese, Nasrullah Memon, Reda Alhajj, Theodor G. Wyeld, Remo Aslak Burkhard, Georges Grinstein, Dennis Groth, Anna Ursyn, Carsten Maple, Anthony Faiola, and Brock Craft, editors, *IV'08: Proceedings of the International Conference on Information Visualisation*, pages 610–615. IEEE Computer Society, 2008. ISBN 9780769532684. doi:10.1109/IV.2008.10.

[NBW06] Mark Newman, Albert-László Barabási, and Duncan J. Watts. *The Structure and Dynamics of Networks*. Princeton Studies in Complexity. Princeton University Press, 2006. ISBN 0691113572.

[NH02] Quang Vinh Nguyen and Mao Lin Huang. A space-optimized tree visualization. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 85–92. IEEE Computer Society, 2002. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173152.

[NH03] Quang Vinh Nguyen and Mao Lin Huang. Space-optimized tree: a connection+enclosure approach for the visualization of large hierarchies. *Information Visualization*, 2(1):3–15, 2003. doi:10.1057/palgrave.ivs.9500031.

[NH05] Quang Vinh Nguyen and Mao Lin Huang. EncCon: an approach to constructing interactive visualization of large hierarchical data. *Information Visualization*, 4(1):1–21, 2005. doi:10.1057/palgrave.ivs.9500087.

[NH06] Matej Novotný and Helwig Hauser. Similarity brushing for exploring multidimensional relations. *Journal of WSCG*, 14(1-3):105–112, 2006.

[NUUT07] Antoine Naud, Shiro Usui, Naonori Ueda, and Tatsuki Taniguchi. Visualization of documents and concepts in neuroinformatics with the 3D-SE viewer. *Frontiers in Neuroinformatics*, 1(7), 2007. doi:10.3389/neuro.11.007.2007.

[ODB06] Richard O'Donnell, Alan Dix, and Linden J. Ball. Exploring the PieTree for representing numerical hierarchical data. In Nick Bryan-Kinns, Ann Blandford, Paul Curzon, and Laurence Nigay, editors, *People and Computers XX – Engage: Proceedings of the Human Computer Interaction 2006*, pages 239–254. Springer, 2006. ISBN 9781846285882. doi:10.1007/978-1-84628-664-3_18.

[ONF07] Benoît Otjacques, Monique Noirhomme, and Fernand Feltz. Innovative visualization tools to monitor scientific cooperative activities. In Yuhua Luo, editor, *CDVE'07: Proceedings of the International Conference on Cooperative Design, Visualization, and Engineering*, Lecture Notes in Computer Science, pages 33–41. Springer, 2007. ISBN 9783540747796. doi: 10.1007/978-3-540-74780-2_4.

[ONG+07] Benoît Otjacques, Monique Noirhomme, Xavier Gobert, Pierre Collin, and Fernand Feltz. Visualizing the activity of a web-based collaborative platform. In *IV'07: Proceedings of the International Conference on Information Visualisation*, pages 251–256. IEEE Computer Society, 2007. ISBN 0769529003. doi:10.1109/IV.2007.137.

[OS08] Krzysztof Onak and Anastasios Sidiropoulos. Circular partitions with applications to visualization and embeddings. In *SCG'08: Proceedings of the Symposium on Computational Geometry*, pages 28–37. ACM Press, 2008. ISBN 9781605580715. doi:10.1145/1377676.1377683.

[PCA02] Helen C. Purchase, David Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, September 2002. doi:10.1023/A:1016344215610.

[PS08] Adam Perer and Ben Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In Margaret Burnett, Maria Francesca Costabile, Tiziana Catarci, Boris de Ruyter, Desney Tan, Mary Czerwinski, and Arnie Lund, editors, *CHI'08: Proceedings of the International Conference on Human Factors in Computing Systems*, pages 265–274. ACM Press, 2008. ISBN 9781605580111. doi: 10.1145/1357054.1357101.

[PTMB09] Harald Piringer, Christian Tominski, Philipp Muigg, and Wolfgang Berger. A multi-threading architecture to support interactive visual exploration. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1113–1120, 2009. doi:10.1109/TVCG.2009.110.

[PvW08] A. Johannes Pretorius and Jarke J. van Wijk. Visual inspection of multivariate graphs. *Computer Graphics Forum*, 27(3):967–974, 2008. doi: 10.1111/j.1467-8659.2008.01231.x.

[PWG05] Bijan Parsia, Taowei Wang, and Jennifer Golbeck. Visualizing web ontologies with cropcircles. In Abraham Bernstein, Ion Androutsopoulos, Duane Degler, and Brian McBride, editors, *Proceedings of the ISWC Workshop on End User Semantic Web Interaction 2005*. CEUR Workshop Proceedings, 2005.

[RA04] Garry Robins and Malcolm Alexander. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computa-*

tional and Mathematical Organization Theory, 10(1):69–94, May 2004. doi:10.1023/B:CMOT.0000032580.12184.c0.

[RC94] Ramana Rao and Stuart K. Card. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In Beth Adelson, Susan T. Dumais, and Judith S. Olson, editors, *CHI'94: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 318–322. ACM Press, 1994. ISBN 0897916506. doi:10.1145/191666.191776.

[RG93] Jun Rekimoto and Mark Green. The information cube: Using transparency in 3D information visualization. In *WITS'93: Proceedings of the Workshop on Information Technology and Systems*, pages 125–132. 1993.

[RH09] René Rosenbaum and Bernd Hamann. Progressive presentation of large hierarchies using treemaps. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnação, Cláudio T. Silva, and Daniel Coming, editors, *Advances in Visual Computing: Proceedings of the International Symposium on Visual Computing 2009*, Lecture Notes in Computer Science, pages 71–80. Springer, 2009. ISBN 9783642105197. doi:10.1007/978-3-642-10520-3_7.

[Ris08] John S. Risch. On the role of metaphor in information visualization. *arXiv.org e-print service*, 0809.0884, 2008.

[RMF09] Sébastien Rufiange, Michael J. McGuffin, and Christopher Fuhrman. Visualisation hybride des liens hiérarchiques incorporant des Treemaps dans une matrice d'adjacence. In Patrick Reignier and Dominique Vaufreydaz, editors, *IHM'09: Proceedings of the International Conference on Association Francophone d'Interaction Homme-Machine*, pages 51–54. ACM Press, 2009. ISBN 9781605584614. doi:10.1145/1629826.1629834.

[Rod05] Peter Rodgers. Graph drawing techniques for geographic visualization. In Jason Dykes, Alan M. MacEachren, and Menno-Jan Kraak, editors, *Exploring Geovisualization*, chapter 7, pages 143–158. Elsevier, 2005.

[RP89] Arnon Rosenthal and Jose A. Pino. A generalized algorithm for centrality problems on trees. *Journal of the ACM*, 36(2):349–361, 1989. doi:10.1145/62044.62051.

[SA06] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006. doi:10.1109/TVCG.2006.166.

[San07] Anders Sandberg. Hilbert tree of life. `http://www.flickr.com/photos/arenamontanus/1916189332/in/`

set-72157594326128194/, November 2007. Retrieved 26-APR-2010.

[SBRG] San Diego Systems Biology Research Group, University of California. BiGG: Database of biochemically, genetically and genomically structured genome-scale metabolic network reconstructions. http://bigg.ucsd.edu. Retrieved 28-APR-2010.

[SCGM00] John Stasko, Richard Catrambone, Marc Guzdial, and Kevin McDonald. An evaluation of space-filling information visualizations for depiction hierarchical structures. *International Journal of Human-Computer Studies*, 53(5):663–694, 2000. doi:10.1006/ijhc.2000.0420.

[Sch04] Matthew C. Schumaker. *Matrix Based Visualization of Graphs*. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY, 2004.

[SCH+06] Xiaohua Sun, Patrick Chiu, Jeffrey Huang, Maribeth Back, and Wolf Polak. Implicit brushing and target snapping: Data exploration and sensemaking on large displays. In Augusto Celentano and Piero Mussio, editors, *AVI'06: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 258–261. ACM Press, 2006. ISBN 1595933530. doi: 10.1145/1133265.1133316.

[SD01] Marc Stamminger and George Drettakis. Interactive sampling and rendering for complex and procedural geometry. In Steven J. Gortler and Karol Myszkowski, editors, *Proceedings of the Eurographics Workshop on Rendering Techniques 2001*, pages 151–162. Springer, 2001. ISBN 3211837094.

[SDW09] Aidan Slingsby, Jason Dykes, and Jo Wood. Configuring hierarchical layouts to address research questions. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):977–984, 2009. doi:10.1109/TVCG.2009.128.

[SGL08] John Stasko, Carsten Görg, and Zhicheng Liu. Jigsaw: supporting investigative analysis through interactive visualization. *Information Visualization*, 7(2):118–132, 2008. doi:10.1057/palgrave.ivs.9500180.

[Shn92] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992. doi:10.1145/102377.115768.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL'96: Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343. IEEE Computer Society, 1996. ISBN 081867508X. doi:10.1109/VL.1996.545307.

[SJOC01] Harvey S. Smallman, Mark St. John, Heather M. Oonk, and Michael B. Cowen. Information availability in 2d and 3d displays. *IEEE Computer*

Graphics and Applications, 21(5):51–57, September-October 2001. doi: 10.1109/38.946631.

[SKW+07] Markus Schedl, Peter Knees, Gerhard Widmer, Klaus Seyerlehner, and Tim Pohle. Browsing the web using stacked three-dimensional sunbursts to visualize term co-occurrences and multimedia content. In *InfoVis'07: Poster Compendium of the IEEE Conference on Information Visualization*, pages 2–3. 2007.

[SLK+09] Marc Streit, Alexander Lex, Michael Kalkusch, Kurt Zatloukal, and Dieter Schmalstieg. Caleydo: Connecting pathways and gene expression. *Bioinformatics*, 2009. doi:10.1093/bioinformatics/btp432. To appear.

[SM07] Zeqian Shen and Kwan-Liu Ma. Path visualization for adjacency matrices. In Ken Museth, Torsten Möller, and Anders Ynnerman, editors, *EuroVis'07: Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 83–90. Eurographics Association, 2007. ISBN 9783905673456. doi:10.2312/VisSym/EuroVis07/083-090.

[Spi03] Jeremy P. Spinrad. *Efficient Graph Representations*. Fields Institute Monographs. American Mathematical Society, 2003. ISBN 0821828150.

[SS98] S. Sarkar and K.N. Sivarajan. Hypergraph models for cellular mobile communication systems. *IEEE Transactions on Vehicular Technology*, 47(2):460–471, May 1998. doi:10.1109/25.669084.

[Stu06] Anthony F. Stuart. Container metaphor for visualization of complex hierarchical data types, United States Patent Application US 2006/0080622 A1, filed Oct. 12, 2004, published Apr. 13, 2006.

[SW01] Ben Shneiderman and Martin Wattenberg. Ordered Treemap layouts. In Keith Andrews, Steven Roth, and Pak Chung Wong, editors, *InfoVis'01: Proceedings of the IEEE Symposium on Information Visualization*, pages 73–78. IEEE Computer Society, 2001. ISBN 0769513425. doi:10.1109/INFVIS.2001.963283.

[SZ00] John Stasko and Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *InfoVis'00: Proceedings of the IEEE Symposium on Information Visualization*, pages 57–65. IEEE Computer Society, 2000. ISBN 0769508049. doi:10.1109/INFVIS.2000.885091.

[TAS09] Christian Tominski, James Abello, and Heidrun Schumann. CGV – an interactive graph visualization system. *Computers and Graphics*, 33(6):660–678, December 2009. doi:10.1016/j.cag.2009.06.002.

[TAvHS06] Christian Tominski, James Abello, Frank van Ham, and Heidrun Schumann. Fisheye tree views and lenses for graph visualization. In Ebad

Banissi, Remo Aslak Burkhard, Anna Ursyn, Jian J. Zhang, Mark Bannatyne, Carsten Maple, Andrew J. Cowell, Gui Yun Tian, and Ming Hou, editors, *IV'06: Proceedings of the International Conference on Information Visualisation*, pages 17–24. IEEE Computer Society, 2006. ISBN 0769526020. doi:10.1109/IV.2006.54.

[TC05] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, 2005. ISBN 0769523234.

[TFS08] Christian Tominski, Georg Fuchs, and Heidrun Schumann. Task-driven color coding. In Ebad Banissi, Liz Stuart, Mikael Jern, Gennady Andrienko, Francis T. Marchese, Nasrullah Memon, Reda Alhajj, Theodor G. Wyeld, Remo Aslak Burkhard, Georges Grinstein, Dennis Groth, Anna Ursyn, Carsten Maple, Anthony Faiola, and Brock Craft, editors, *IV'08: Proceedings of the International Conference on Information Visualisation*, pages 373–380. IEEE Computer Society, 2008. ISBN 9780769532684. doi:10.1109/IV.2008.24.

[TJ92] David Turo and Brian Johnson. Improving the visualization of hierarchies with treemaps: Design issues and experimentation. In Arie Kaufman and Gregory M. Nielson, editors, *Visualization'92: Proceedings of the IEEE Conference on Visualization*, pages 124–131. IEEE Computer Society, 1992. ISBN 0818628979. doi:10.1109/VISUAL.1992.235217.

[TK09] Jim Thomas and Joe Kielman. Challenges for visual analytics. *Information Visualization*, 8(4):309–314, 2009. doi:10.1057/ivs.2009.26.

[TL01] Monica Tavanti and Mats Lind. 2D vs. 3D, implications on spatial memory. In Keith Andrews, Steven Roth, and Pak Chung Wong, editors, *InfoVis'01: Proceedings of the IEEE Symposium on Information Visualization*, pages 139–146. IEEE Computer Society, 2001. ISBN 0769513425. doi:INFVIS.2001.963291.

[TM02] Soon Tee Teoh and Kwan-Liu Ma. RINGS: A technique for visualizing large hierarchies. In Michael T. Goodrich and Stephen G. Kobourov, editors, *GD'02: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 51–73. Springer, 2002. ISBN 9783540001584. doi:10.1007/3-540-36151-0_25.

[TON03] Yoichi Tanaka, Yoshihiro Okada, and Koichi Niijima. Treecube: Visualization tool for browsing 3D multimedia data. In *IV'03: Proceedings of the International Conference on Information Visualisation*, pages 427–432. IEEE Computer Society, 2003. ISBN 0769519881. doi:10.1109/IV.2003.1218020.

[TON04] Yoichi Tanaka, Yoshihiro Okada, and Koichi Niijima. Interactive interfaces of treecube for browsing 3D multimedia data. In Maria Francesca Costabile, editor, *AVI'04: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 298–302. ACM Press, 2004. ISBN 1581138679. doi:10.1145/989863.989914.

[TS07] Ying Tu and Han-Wei Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1286–1293, November-December 2007. doi:10.1109/TVCG.2007.70529.

[TSWS08] Christian Tominski, Petra Schulze-Wollgast, and Heidrun Schumann. Visual methods for analyzing human health data. In Nilmini Wickramasinghe and Eliezer Geisler, editors, *Encyclopedia of Healthcare Information Systems*, pages 1357–1364. IGI Global, 2008. ISBN 9781599048895.

[TTT07] Vassilis Tsiaras, Sofia Triantafilou, and Ioannis G. Tollis. Treemaps for directed acyclic graphs. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *GD'07: Proceedings of the International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pages 377–388. Springer, 2007. ISBN 9783540775362. doi:10.1007/978-3-540-77537-9_37.

[Tuf90] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990. ISBN 0961392118.

[Tuf01] Edward Rolf Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition edition, 2001. ISBN 0961392142.

[Tun99] Daniel Tunkelang. *A Numerical Optimization Approach to General Graph Drawing*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, January 1999.

[TvW02] Alexandru Telea and Jarke J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In David S. Ebert, Pere Brunet, and Isabel Navaz, editors, *VisSym'02: Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 251–260. Eurographics Association, 2002. ISBN 158113536X.

[TZB96] Oleg N. Temkin, Andrew V. Zeigarnik, and Danail Bonchev. *Chemical Reaction Networks: A Graph-Theoretical Approach*. CRC Press, 1996. ISBN 0849328675.

[UEJ+07] Adelinde M. Uhrmacher, Roland Ewald, Mathias John, Carsten Maus, Matthias Jeschke, and Susanne Biermann. Combining micro and macro-modeling in DEVS for computational biology. In Shane G. Henderson, Bahar Biller, Ming-Hua Hsieh, John Shortle, Jeffrey D. Tew, and Russell R.

Barton, editors, *WSC'07: Proceedings of the Winter Simulation Conference*, pages 871–880. IEEE Computer Society, 2007. ISBN 1424413060. doi:10.1109/WSC.2007.4419683.

[Ung10] Andrea Unger. *Visual Support for the Modeling and Simulation of Cell Biological Processes*. Ph.D. thesis, Universität Rostock, 2010.

[vH03] Frank van Ham. Using multilevel call matrices in large software projects. In Tamara Munzner and Stephen North, editors, *InfoVis'03: Proceedings of the IEEE Symposium on Information Visualization*, pages 29–34. IEEE Computer Society, 2003. ISBN 0780381548. doi:10.1109/INFVIS.2003.1249030.

[vH08] Frank van Ham. *Interactive Visualization of Large Networks: Technical Challenges and practical applications*. Verlag Dr. Müller, 2008. ISBN 9783639107005.

[vHvW02] Frank van Ham and Jarke J. van Wijk. Beamtrees: Compact visualization of large hierarchies. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 93–100. IEEE Computer Society, 2002. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173153.

[vHvW04] Frank van Ham and Jarke J. van Wijk. Interactive visualization of small world graphs. In Matt Ward and Tamara Munzner, editors, *InfoVis'04: Proceedings of the IEEE Symposium on Information Visualization*, pages 199–206. IEEE Computer Society, 2004. ISBN 0780387791. doi:10.1109/INFVIS.2004.43.

[vLdL03] Robert van Liere and Wim de Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):206–212, April-June 2003. doi:10.1109/TVCG.2003.1196007.

[vN79] Cyriel van Nuffelen. Enumeration techniques in directed hypergraphs. In K. Iracki, Kazimierz Malanowski, and Stanisaw Walukiewicz, editors, *Proceedings of the IFIP Conference on Optimization Techniques 1979*, Lecture Notes in Control and Information Sciences, pages 330–333. Springer, 1979. ISBN 9783540100812. doi:10.1007/BFb0006620.

[Voi01] Denny Voigt. *WWW-basierte Darstellung komplexer Informationsstrukturen*. Master's thesis, University of Rostock, July 2001.

[VvWvdL06] Roel Vliegen, Jarke J. van Wijk, and Erik-Jan van der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789–796, September-October 2006. doi:10.1109/TVCG.2006.200.

[vWvdW99] Jarke J. van Wijk and Huub van de Wetering. Cushion Treemaps: Visualization of hierarchical information. In Daniel Keim and Graham Wills, editors, *InfoVis'99: Proceedings of the IEEE Symposium on Information Visualization*, pages 73–78. IEEE Computer Society, 1999. ISBN 0769504310. doi:10.1109/INFVIS.1999.801860.

[VWvH+07] Fernanda B. Viégas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. Many eyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, November-December 2007. doi:10.1109/TVCG.2007.70577.

[Wal90] John Q. Walker, II. A node-positioning algorithm for general trees. *Software – Practice and Experience*, 20(7):685–705, July 1990. doi: 10.1002/spe.4380200705.

[Wat99] Martin Wattenberg. Visualizing the stock market. In *CHI'99: Extended abstracts of the SIGCHI conference on Human Factors in Computing Systems*, pages 188–189. ACM Press, 1999. ISBN 1581131585. doi: 10.1145/632716.632834.

[Wat02] Martin Wattenberg. Arc diagrams: visualizing structure in strings. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 110–116. IEEE Computer Society, 2002. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173155.

[Wat05] Martin Wattenberg. A note on space-filling visualizations and space-filling curves. In John Stasko and Matt Ward, editors, *InfoVis'05: Proceedings of the IEEE Symposium on Information Visualization*, pages 181–185. IEEE Computer Society, 2005. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532145.

[WC04] Bang Ye Wu and Kun-Mao Chao. *Spanning Trees and Optimization Problems*. Chapman and Hall, 2004. ISBN 1584884363.

[WD08] Jo Wood and Jason Dykes. Spatially ordered treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1348–1355, 2008. doi: 10.1109/TVCG.2008.165.

[Wet03] Kai Wetzel. Pebbles – using circular treemaps to visualize disk usage. `http://lip.sourceforge.net/ctreemap.html`, 2003. Retrieved 26-APR-2010.

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. ISBN 0521387078.

[WFC+06] Pak Chung Wong, Harlan Foote, George Chin, Jr., Patrick Mackey, and Ken Perrine. Graph signatures for visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1399–1413, November/December 2006. doi:10.1109/TVCG.2006.92.

[WFK+02] Pak Chung Wong, Harlan Foote, David L Kao, Ruby Leung, and Jim Thomas. Multivariate visualization with data fusion. *Information Visualization*, 1(3-4):182–193, December 2002. doi:10.1057/palgrave.ivs.9500024.

[Wil96] Graham J. Wills. Selection: 524,288 ways to say "this is interesting". In Nahum D. Gershon, Stuart Card, and Stephen G. Eick, editors, *InfoVis'96: Proceedings of the IEEE Symposium on Information Visualization*, pages 54–60. IEEE Computer Society, 1996. ISBN 081867668X. doi:10.1109/INFVIS.1996.559216.

[Wil05] Leland Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer, 2nd edition, 2005. ISBN 0387245448. doi:10.1007/0-387-28695-0.

[WL07] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In Jonathan I. Maletic, Alexandru Telea, and Andrian Marcus, editors, *VisSoft'07: Proceedings of the IEEE International Workshop onVisualizing Software for Understanding and Analysis*, pages 92–99. IEEE Computer Society, 2007. ISBN 1424406005. doi:10.1109/VISSOF.2007.4290706.

[WLSS09] Manuela Waldner, Alexander Lex, Marc Streit, and Dieter Schmalstieg. Design considerations for collaborative information workspaces in multi-display environments. In *CoVIS'09:Proceedings of the Workshop on Collaborative Visualization on Interactive Surfaces*. 2009.

[WP06] Taowei David Wang and Bijan Parsia. CropCircles: Topology sensitive visualization of OWL class hierarchies. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora Aroyo, editors, *ISWC'06: Proceedings of the International Semantic Web Conference*, Lecture Notes in Computer Science, pages 695–708. Springer, 2006. ISBN 9783540490296. doi:10.1007/11926078_50.

[WTM06] Yue Wang, Soon Tee Teoh, and Kwan-Liu Ma. Evaluating the effectiveness of tree visualization systems for knowledge discovery. In Beatriz Sousa Santos, Thomas Ertl, and Kenneth I. Joy, editors, *EuroVis'06: Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 67–74. Eurographics Association, 2006. ISBN 3905673312. doi:10.2312/VisSym/EuroVis06/067-074.

[WWDW06] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. Visualization of large hierarchical data by circle packing. In Rebecca Grinter,

Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson, editors, *CHI'06: Proceedings of the International Conference on Human Factors in Computing Systems*, pages 517–520. ACM Press, 2006. ISBN 1595933727. doi:10.1145/1124772.1124851.

[YWR02] Jing Yang, Matthew O. Ward, and Elke A. Rundensteiner. InterRing: An interactive tool for visually navigating and manipulating hierarchical structures. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 77–84. IEEE Computer Society, 2002. ISBN 076951751X. doi: 10.1109/INFVIS.2002.1173151.

[ZG02] Jin Zhang and Jianya Gong. Hypergraph-based object-oriented model and hypergraph theory for GIS. *Geo-Spatial Information Science*, 5(1):37–43, March 2002. doi:10.1007/BF02863493.

[ZK08] Caroline Ziemkiewicz and Robert Kosara. The shaping of information by visual metaphors. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1269–1276, 2008. doi:10.1109/TVCG.2008.171.

[ZMC05] Shengdong Zhao, Michael J. McGuffin, and Mark H. Chignell. Elastic hierarchies: combining Treemaps and node-link diagrams. In John Stasko and Matt Ward, editors, *InfoVis'05: Proceedings of the IEEE Symposium on Information Visualization*, pages 57–64. IEEE Computer Society, 2005. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532129.

Thesis Statements

1. Network structures (graphs) have become a natural part of everyday life. Their analysis gives insight in many aspects of the real-world facets they are capturing, for instance most connected actors in social networks. A fundamental part of this analysis is the exploration of graphs, in order to discover unexpected patterns or properties, as opposed to confirming expected ones. Methods for detecting such patterns without preconceived knowledge about them are a current research topic.

2. To bring the users and their nondescript notion of unexpectedness into the analysis process, interactive graph visualization methods to support the exploration are in widespread use. This introduces an additional layer between the graph data and the analysis tasks to be performed on it, which benefits the exploration on one side, but introduces additional challenges to the visualization design process on the other side.

3. Explorative graph visualization faces design challenges on all three levels involved: on *data level* in terms of large graph sizes and of different graph types, on *representation level* when ensuring meaningful and readable, yet space-efficient layouts, and on *task level*, which is variable and neither purely visual nor purely explorative.

4. In order to support the development of concrete solution approaches to these challenges, their detailed and interwoven discussion is a necessary step to consolidate the known approaches and identify recurring design strategies. Only this allows to recognize open problems and to derive highly customized explorative graph visualizations geared towards their solution.

5. On the level of *representation*, the discussion of how to choose a meaningful and readable visualization from the wealth of all possible techniques is lead for the subset of implicit tree visualizations. The design space spanned for this class of visualizations is complete and consistent in the sense that it covers all known and unknown implicit tree visualizations and only implicit tree visualizations. It serves to communicate the problems in implicit visualization design and aids in interactively trying out new visualizations. The applicability of the design space as a means to browse the huge number of possible implicit tree visualizations and create custom visualization techniques is underlined by a number of novel designs.

6. On *data* level, the challenge of visualizing large graphs leads automatically to considerations of space and runtime efficiency, whereas the different graph types present a challenge for a visualization's expressiveness. In the examples given, a recursive layout refinement for trees and a layered approach for bipartite graphs are

used – both being fast to layout and compact in their display. The applicability of both design strategies is shown through two concrete instantiations, a point-based tree visualization and a table-based visualization for bipartite graphs and hypergraphs.

7. At the basis of all discussion on the *task* level lies the observation that exploratory visual analysis tasks are part of a comprehensive analysis workflow, which includes computational tasks and confirmatory tasks, as it is also recognized by current research in visual analytics. Hence, supporting the design of the combined graph visualization and computation, a conceptual visual analytics framework is devised that utilizes initially gathered information about the input graph to adjust the generic exploration workflow to the specifics of the given graph. This does also include the computation of exploratory, confirmatory, and even combined measures to be subsequently visualized, as it is demonstrated for a use case from the domain of social network analysis. In case of multiple data sets to be explored, the analysis workflow is used to automatically derive suggestions for concrete analysis paths, including the data sets and views needed in their course. This is achieved by a comprehensive modeling of all three levels of a concrete application scenario – in this case of a biomedical setup.

8. Most of the discussed solution approaches and their concrete instantiations have been developed in close collaboration within the graduate school “dIEM oSiRiS” – either for use in the field of Modeling and Simulation, or for the use in the field of Biomedicine.

Resume

Personal Data

Name	Hans-Jörg Schulz
Place of Living	Bad Doberan
Date of Birth	26-February-1979
Place of Birth	Rostock
Nationality	German

Education

09/2008-04/2010 and 11/2006-05/2008	PhD student at the interdisciplinary research training school *GRK dIEM oSiRiS*, University of Rostock (funded by the *German Research Foundation DFG*)
06/2008-08/2008	research internship at the *Visual Communication Lab* of the *IBM T.J. Watson Research Center*, Cambridge, MA, USA
10/2007-11/2007 and 02/2007-03/2007	research visit at the *DIMACS Center for Discrete Mathematics and Theoretical Computer Science*, Rutgers University, NJ, USA (funded by *DIMACS* and *DyDAn*)
07/2005-10/2006	PhD student at the *Department of Computer Graphics*, University of Rostock (funded by the German protestant church's scholarship program *Evangelisches Studienwerk e.V. Villigst*)
06/2005-07/2005	research staff member of the Mobile Multimedia Research Focus Project of the State (*Landesforschungsschwerpunkt M6C*) at the University of Rostock
11/2004-05/2005	internship at the *ECS GmbH* in Rostock and Neumarkt as software developer
10/2004	university diploma (Dipl.-Inf.) from the University of Rostock
06/1998	high school diploma (Abitur) from the *Jugenddorf Christophorus Gymnasium* in Rostock

Erklärung

Ich, Hans-Jörg Schulz, erkläre, dass ich die vorgelegte Dissertationsschrift mit dem Thema: Explorative Graph Visualization selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, ohne die (unzulässige) Hilfe Dritter verfasst und auch in Teilen keine Kopien anderer Arbeiten dargestellt habe.

Datum Unterschrift

4. Juli 2010